SH20-1773-0

Systems

IBM System/370 Special Real Time Operating System Programming RPQ Z06751 Description and Operation Manual

Program Number 5799-AHE

The Special Real Time Operating System Programming RPQ is a system which augments the services provided by OS/VS1 to support realtime computer operations. The Special Real Time Operating System is designed to meet the needs of Electric Utility Energy Management Systems and oil refinery applications, but is not restricted to these applications. The Special Real Time Operating System runs as an OS/VS1 job step and performs services which support independent task management, time management, and data base management. The installation of the Special Real Time Operating System on an OS/VS1 system requires no modifications to the OS/VS1 System.

This manual contains all the information necessary to understand, install, use, and operate the Special Real Time Operating System PRPQ.

This Programming RPQ is available on a special quotation only (see inside front cover).

IBM

First Edition (January 1976)

This edition applies to Version 1, Modification 0 of the Special Real Time Operating System, PRPQ (Programming Request for Price Quotation) number:

5799-AHE, Version 1, Modification 0

and to all subsequent versions and modifications until otherwise indicated in new editions or technical newsletters.

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest *System/370 Bibliography* (GA22-6822) for the editions that are applicable and current.

The PRPQ described in this manaul, and all licensed materials available for it, are provided by IBM on a special quotation basis only, under the terms of the License Agreement for IBM Program Products. Your local IBM branch office can advise you regarding the special quotation and ordering procedures.

A form for reader's comments is provided at the back of this publication. If the form has been removed, send your comments to IBM Corporation, General Systems Division, Dept. 20GB, 2800 Sand Hill Road, Menlo Park, California 94025. Comments become the property of IBM.

CONTENTS

Section Page

LIST OF FIGURES

PREFACE

This publication provides information on the Special Real Time Operating System (5799-AHE).

This manual is organized so that it can be used as four separate manuals, each chapter addressing the needs of a different audience. In each case, the intended audience is a group within a typical computer department. The intended audience and the applicable chapters are:

- Management -- Chapter 1 - GENERAL INFORMATION
- Application Programmers -- Chapter 2 - APPLICATION SERVICES
- System Programmers -- Chapter 3 - INSTALLATION GUIDE
- Operators -- Chapter 4 - OPERATORS' REFERENCE

The intended audience for the section entitled "GENERAL INFORMATION" includes those people wishing to gain an overview of the Special Real Time Operating System and to become familiar with the general functions of the PRPQ. This chapter is prerequisite reading to the following chapters.

The section entitled "APPLICATION SERVICES" is intended to be used by programmers to gain knowledge of the realtime system concepts and processing methods. It is technically oriented. Users of this section should have a thorough knowledge of programming techniques as well as a general knowledge of Operating System/Virtual Storage (OS/VS1). The parts of this section dealing with high-level language interface require a prior knowledge of the language specifications for the given high-level language.

The intended audience for the section entitled "INSTALLATION GUIDE" are the people involved with the preparation for and the installation of the Special Real Time Operating System PRPQ. Users of this section should have prerequisite knowledge of OS/VS1 system programming, job control (JCL), SYSGEN, and generally a thorough knowledge of OS/VS1.

The final section entitled "OPERATORS' REFERENCE" is intended for the system console operator. This section contains operations information to enable the operator to start, terminate, and communicate with the Special Real Time Operating System. The operator should be familiar with OS/VS1 operating techniques.

INTRODUCTION

The Special Real Time Operating System PRPQ is a support program that augments the services of OS/VS1 to support realtime applications and provides a stable operating environment. The services provided by OS/VS1 are still available to a program or system of programs utilizing the Special Real Time Operating System. Although in some cases, the Special Real Time Operating System acts as an interface between OS/VS1 and user programs, as shown in Figure 1-1.

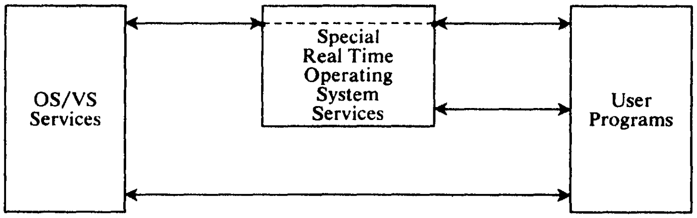


Figure 1-1. User Special Real Time Operating System-OS/VS Interface

The installation of the Special Real Time Operating System on the user's OS/VS1 system entails no modifications to the OS/VS1 system; although there are certain additions to that system. In particular, there are supervisor call (SVC) routines that must be included into the OS/VS1 libraries. The Special Real Time Operating System services augment the OS/VS1 services in the following areas:

- Lower overhead through independent task management
- Significantly enhanced time management routines
- Realtime message handler
- Data base management and data base logging
- Duplicate data set support for critical Special Real Time Operating System and user data sets.
- Selective termination of units of work
- Selective data recording for post-run analysis
- Input message processing
- High-level language support for PL/I and FORTRAN
- Failover restart support.

In addition to these enhancements, the Special Real Time Operating System is designed so that each user builds and tailors his own Special Real Time Operating System for his own equipment configuration and for his own operational requirements through a system build or system generation (SYSGEN) process.

Creation and modification of the table structure and initial conditions for the online system are handled by offline utility programs. As a result, changes in this area do not require additional system generations.

GENERAL DESCRIPTION

The Special Real Time Operating System is designed to enhance areas which are critical to a realtime operation. The following paragraphs discuss the enhancements which are provided by the Special Real Time Operating System.

Independent task management allows a task to be created and remain in existence when its processing is finished. Units of work are queued to the task, and the task does its processing with the overhead of resource allocation, initiation, and termination only once and not for any subsequent processing of units of work by the task. The Special Real Time Operating System task management routines bring a task into virtual storage, queue work against the task, and delete the task or specified units of work upon request from the user. This results in a significant decrease in task management processing overhead. Also, the Special Real Time Operating System provides the user with greater flexibility and control over the work to be processed by a given task.

The Special Real Time Operating System time management services fall into two categories. First, the Special Real Time Operating System maintains system time and date independently of OS/VS1 time and date. The Special Real Time Operating System time can be synchronized with an external time source or can be adjusted by manual inputs. Second, the Special Real Time Operating System time management services provide the user with the capability to pass a work request to a specific task at a selected time and, optionally, have the work request repeated at a specified interval.

The realtime message handler allows messages which the user has previously defined offline to be accessed in realtime. These messages can then be selected by message number, modified, and routed in realtime with minimum impact on system performance.

The Special Real Time Operating System data base services maintain a data base in virtual storage and on direct access storage. The services also allow the data base to be accessed independently by several tasks. The data is defined as a group of named arrays and named items within the arrays. The data is accessed by name, and this allows associated programs to be coded independently of most changes or additions to the data base. The content of the data base arrays may be logged to history files on a cyclic or demand basis. The logged data can then be used for reinitialization of the data base after a system outage as well as a historical record of system operation. The data base arrays and items are created by an offline utility program for use in the realtime run.

Duplicate copies of the critical Special Real Time Operating System and/or user data sets can be maintained to provide backup copies should the primary copy experience a failure. This provides a smooth transition when making modifications to these critical data sets. The duplicate data set support services are optional and may be selected when the Special Real Time Operating System is created. Duplicate data sets may be used to keep backup copies of the data base data sets.

The impact of failing tasks is minimized through selective termination of units of work. If a task experiences a failure while executing a unit of work, that unit of work is terminated. However, the task is maintained, and all remaining units of work queued to the task will be executed.

The Special Real Time Operating System record and playback feature provides services for the user to define data that can be recorded on tape or direct access device during realtime execution. This recorded data can then be used for post-run analysis or as test data on a subsequent program execution.

An input message processor is provided to allow for operator communication. This allows operator commands to be entered through a system console and routed to designated user programs.

An interface is provided so that the user may code his programs in PL/I or FORTRAN and request the normal Special Real Time Operating System services through the interface program.

The Special Real Time Operating System has facilities to allow execution on a two CPU configuration where a job in the backup CPU monitors the performance of the online CPU. When either CPU recognizes that a failure has occurred, that CPU can request a failover, and the backup CPU becomes the online CPU. Failover can also be initiated by program request to facilitate scheduled maintenance or changes to the operational environment.

CUSTOMER RESPONSIBILITES

It is the customer's responsibility to provide in his installation:

- Facilities and minimum hardware configuration required for the Special Real Time Operating System
- Ordering, generation, and testing of the host OS/VS1 system
- Ordering, generation, and testing of the Special Real Time Operating System
- Processing programs required for the realtime operation
- Ordering, generation, and testing of any related PRPQs or program products to be installed
- Data set contents for defining initial values, limits, and other control parameters
- A thorough knowledge of his system and his desired control strategy
- Orders for required computer and terminal equipment needed in the system
- Installation of any instrumentation and/or common carrier facilities required to meet his desired control strategy
- Design and implementation of any specialized application programs and/or display formats required to meet his control strategy
- Training of personnel

PROGRAMMING SYSTEMS

All Special Real Time Operating System programs are coded using the System/370 Assembler Language. The Special Real Time Operating System executes under control of IBM Operating System/Virtual Storage 1, Version 3.0 or a later release. The following components of OS/VS1 are required:

- Supervisor
- Sequential Access Method
- Direct Access Method
- Linkage Editor
- Loader
- System Assembler
- System Utilities
- Partitioned Access Methods.

In addition to the OS/VS1 components, the user may require any of the following:

- PL/I F (360S-NL-511) and PL/I F Subroutine Library (360S-LM-512 VS1)
- PL/I Optimizing Compiler - 5734-PL1
- PL/I Optimizing Compiler and Libraries - 5734 - PL3
- PL/I Resident Library - 5734-LM4
- PL/I Transient Library - 5734-LM5
- FORTRAN IV (G1) - 5734-F02
- FORTRAN IV Library - 5734-LM1
- FORTRAN IV (H Extended) - 5734-F03
- FORTRAN IV Library - 5734-LM3.

SYSTEM CONFIGURATION

The following minimum configuration is required to compile and execute the Special Real Time Operating System.

The machine configuration for the Special Real Time Operating System varies according to the user's application needs. Typical systems are shown as a guideline:

- For Compilation - A 3135 Processing Unit Model DH (245,760 bytes) and appropriate system console. Sufficient Input/Output (I/O) devices must be included to support the requirements for system input, system output, system residence, and system data sets.
- Minimum Operational System - A 3135 Processing Unit Model H (245,760 bytes) including one byte multiplexer channel, one block multiplexer channel, and floating-point instruction set. The configuration must include sufficient I/O devices to support the requirements for system output, system residence, and system data sets. Sufficient direct access storage must be provided to satisfy user information storage requirements. Direct access devices may be chosen from a 2305 Fixed Head Storage, a 2319 Disk Storage Control (Integrated), a 3330 (3333) Disk Storage Facility, 3340 Disk Storage Facility, or combinations.

A magnetic tape unit (9-track) must be available for program distribution and maintenance.

Storage requirements for the Special Real Time Operating System are presented below. The figures are approximate and assume a typical customer environment. They are intended as a guide only.

STORAGE REQUIREMENTS

Figure 1-2 shows the approximate Virtual Storage required by the load modules which comprise the Special Real Time Operating System. The total size represents the approximate maximum number of bytes of storage required for all load modules of each function. Several functions are selectable by Special Real Time Operating System SYSGEN which may reduce the total size of any SYSGENed system from these values. The table includes estimates for routines which are used in an offline environment only and will never be a part of the online system. Some of the routines may be a part of the online system during initialization for a short duration when requested by the user or while processing unusual conditions.

The frequently used column represents the approximate number of bytes of each function which may be used frequently in most systems during a continuing realtime execution. The actual use of any function is dependent upon the application programs and as such, the amount of virtual or real storage occupied by any function is predictable only through analysis of the application.

In addition to the storage represented in Table 1, approximately 320 bytes are added to the OS/VS1 fixed nucleus, and 7700 bytes are added to the pageable nucleus.

The Special Real Time Operating System programs also require approximately five cylinders of a 3330 direct access storage device (or equivalent).

These figures do not include virtual storage or direct access storage which are required for the user's data base.

Function	Frequently Used	Total Size
Task Management	5,000	11,000
Time Management	3,000	5,000
Data Base	4,000	5,000
Data Base Logging		6,000
Message Handler	3,300	3,300
Data Recording		7,000
Report Data Output		900
Duplicate Data Set Support	5,000	22,000
Input Message Processing		7,400
System Initialization		41,000
Failover/Restart	1,000	20,000
FORTRAN PL/I Interface		2,000
Offline Utility Routines		35,000

*Specifies functions wich are optionally selected by the user when he generates his Special Real-Time Operating System.

Figure 1-2. Storage Requirements

TIMING INFORMATION

The timing information given here is meant to aid the user in evaluating factors which may impact the performance of the Special Real Time Operating System. Timings were obtained on a Release 3.0 version of OS/VS1 with eight megabytes of virtual storage and System Management Facility (SMF). The following was the basic hardware configuration:

- System/370 Model 145
- 512K bytes of main storage
- Four 3330 direct access storage devices.

While timing statistics were being gathered, no other jobs were executing. The Special Real Time Operating System was generated with the following options:

- Two-partition support
- Duplicate data set support
- Failover/restart.

The test data base consisted of 46 arrays. Of these arrays, 5 were loggable and 12 were direct access storage resident arrays (this includes 5 log arrays). Of the loggable arrays, 4 were refreshed during initialization.

The following chart gives approximate timings for the major Special Real Time Operating System services. The timings all include OS/VS1 control program services. Task management timings do not include the time of execution of the test program. Times are given as CPU time and as such do not represent elapsed time. The elapsed time could vary greatly depending on system activity, paging, I/O activity, device types, etc.

Caution and judgment should be used in evaluating these statistics due to the many OS/VS1 SYSGEN options and other variables involved. The statistics must be interpreted only as the results obtained in the environment described, and not as a commitment to be met in any or all environments. All times are shown in millisecond units (ms).

TIMING CHART

INITIALIZATION

Basic Jobstep Initialization (includes task management initialization)	900-1200 ms
Time Management Initialization	125-150 ms
Data Base Initialization*	1500-up ms
Logging Initialization ** (includes data base refresh)	425-up ms
Supplementary Services *** (includes Message Handler & Duplicate Data Set Support)	900-up ms

TASK MANAGEMENT

PATCH to existing independent task for a reentrant load module previously loaded	3.70-5.0 ms
PATCH to existing independent task for a reentrant load module not previously loaded	60-100 ms
PATCH to existing independent task for a non-reentrant load module	50-100 ms
PATCH to dependent task (or non-existing independent task) for a reentrant load module previously loaded, assuming there were no dormant Special Real Time Operating System tasks available	25-75 ms
PATCH to dependent task (or non-existing independent task) for a reentrant load module previously loaded, assuming a dormant Special Real Time Operating System task is available	20-75 ms
PATCH to dependent task (or non-existing independent task) for a reentrant load module not previously loaded, assuming a dormant Special Real Time Operating System task is available	65-100 ms
PATCH to dependent task (or non-existing independent task) for a reentrant load module not previously loaded, assuming there were no dormant Special Real Time Operating System task available	70-150 ms
PATCH to dependent task (or non-existing independent task) for a non-reentrant load module, assuming there were no	75-155 ms

*Dependent upon size of data base.

**Dependent upon number of log arrays and initialization refresh options.

***Dependent upon number of messages and the number of duplicate data sets.

dormant Special Real Time Operating System tasks available	
PATCH to dependent task (or non-existing independent task) for a non-reentrant load module, assuming a dormant Special Real Time Operating System task is available	70-150 ms
REPATCH SVC	1-5 ms
DPATCH SVC	6-2.5 ms
TIME MANAGEMENT	
TIME - update time array routine	3-5 ms
PTIM - execute PATCH routine	6-15 ms
PTIME SVC	5-10 ms
DATA BASE*	
GETARRAY/PUTARRAY	
TYPE=ADDR	2.5-5.0 ms
TYPE=SPEC	15-40 ms
TYPE=DATA	2.75-5.0 ms
GETITEM/PUTITEM	
TYPE=ADDR	15.0-55.0 ms
TYPE=SPEC	15.0-55.0 ms
TYPE=DATA(address given)	3.0-5.0 ms
TYPE=DATA(no address given)	15.0-55.0 ms
GETBLOCK/PUTBLOCK	
VS resident	2.0-3.5 ms
DA resident	10-20 ms
DATA BASE LOGGING**	
PUTLOG	
NORMAL	13.0-40.0 ms
LOGHDR	22.0-65.0 ms
BLKLST	10.0-30.0 ms
GETLOG	14.0-100 ms
DUMPLOG	200-up ms
SUPPLEMENTARY SERVICES	
CHAIN	0.5-1.5 ms
DEFLOCK	2.5-5.0 ms
LOCK	0.1-0.5 ms
GETWA	0.45-1.0 ms
MESSAGE HANDLER(includes PATCH)	30-50 ms
DUPLICATE DATA SET	
DDS READ/DDSWRITE	12-50 ms
DDS CHECK	7-25 ms
DDSPOINT/DDSFIND	5-20 ms
DDSBLDL	20-60 ms

*Dependent upon size of data base and number of ITEMS being processed.

**Dependent upon number and size of log arrays and number of log copies.

CHAPTER 2. APPLICATION SERVICES

INTRODUCTION

The objectives of the Special Real Time Operating System in a real-time environment are to provide additional services to user coded, real-time programs and to minimize the impact normally caused by ABENDing programs. The additional services are provided for lower supervisor overhead and added capabilities and flexibility in the areas of task management, time management, data base, message handling, and failover restart, as well as other less significant enhancements. Minimizing system impact due to ABENDs is accomplished by isolating user tasks from one another and by handling work requests as separate entities from the user programs.

GENERAL DESCRIPTION

The Special Real Time Operating System, by itself, as a real-time program, does meaningful processing only when its services are requested by user programs in a real-time environment. The Special Real Time Operating System services are requested through the use of macro calls which invoke the Special Real Time Operating System SVC routines or branch to the Special Real Time Operating System subroutines. This is shown in Figure 2-1.

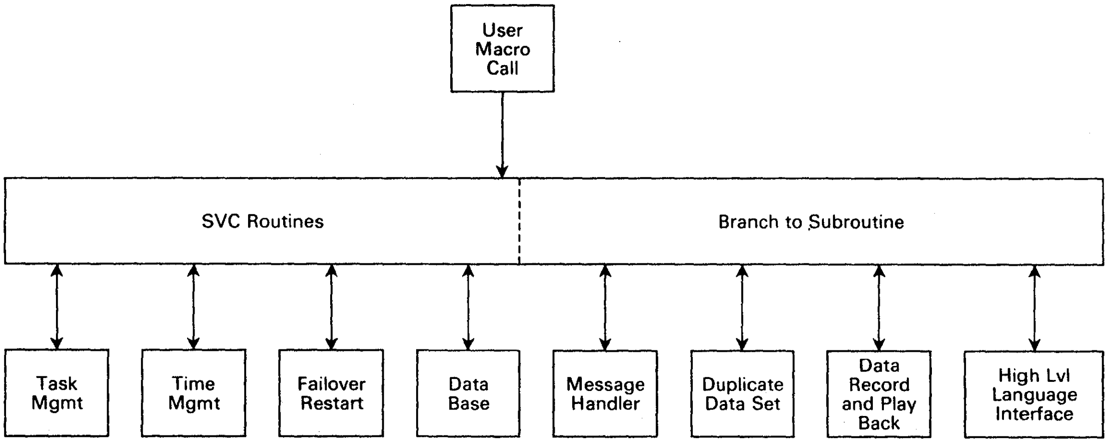


Figure 2-1. The Special Real Time Operating System Overview of the Online System

Figure 2-1 shows the major areas in which the Special Real Time Operating System supplies services for real-time execution

The task management services provide facilities to create the real-time task, queue work to an existing task, or delete a task. These services are provided to the user through the PATCH, REPATCH, and DPATCH macros.

Time management services allow for maintenance of time and for causing work to be passed to tasks at a given time or cyclically for a given interval. The time management services are available to the user via the PTIME macro.

For a real-time environment, the system must have the ability to recover quickly from a failure or system outage. The Special Real Time

Operating System failover restart services allow for a fast switch to a backup-CPU (failover) or a fast restart in the failing CPU. These services are either automatic or under operator control. The data base services in the real-time application allow the user to access the data base but prevent (when requested) access to data by one program if that data is currently being modified by another program. User access to the data base is achieved through six macros; GETITEM, PUTITEM, GETBLOCK, PUTBLOCK, GETARRAY, and PUTARRAY.

The data base, or portions of it, may be logged at given intervals to create a history file. The user interface to the logging routines is through the GETLOG, PUTLOG, and DUMPLOG macros.

The real-time message handler provides a service whereby predefined messages may be retrieved, modified, and routed to predefined devices in real-time. The user interface to this service is through the MESSAGE macro.

Duplicate data set support provides a service whereby the user can maintain duplicate copies of critical data sets. The user requests this service via the DDSBLDL, DDSCLOSE DDSDCB, DDSFIND, DDSOPEN, and DDSSTOW macros.

Data record and playback provide a facility for the user to record areas of virtual storage under program control and later to retrieve or play back the data. The user requests data to be recorded via the RECORD macro.

The high-level language interface programs provide an interface for the real-time services to be used from a PL/I or FORTRAN program.

Each of the Special Real Time Operating System services shown in Figure 2-1 is described in detail in the following sections. Additional services are described later. For the convenience of the application programmer, all online macros are described in detail in the section entitled 'Special Real Time Operating System Online Macros'. The macros in this section are listed in alphabetical sequence.

PROCESSING DESCRIPTION

TASK MANAGEMENT

The Special Real Time Operating System task management services are an extension of the OS/VS1 task supervision and virtual storage supervision to make more efficient use of system resources in a real-time processing system. These additional services are provided by the Special Real Time Operating System through the use of SVC routines, monitor routines, and service subroutines. This is shown in Figure 2-2. The service subroutines can be used only by the SVC routines and the monitor routines. The user invokes the monitor through the SVC interface.

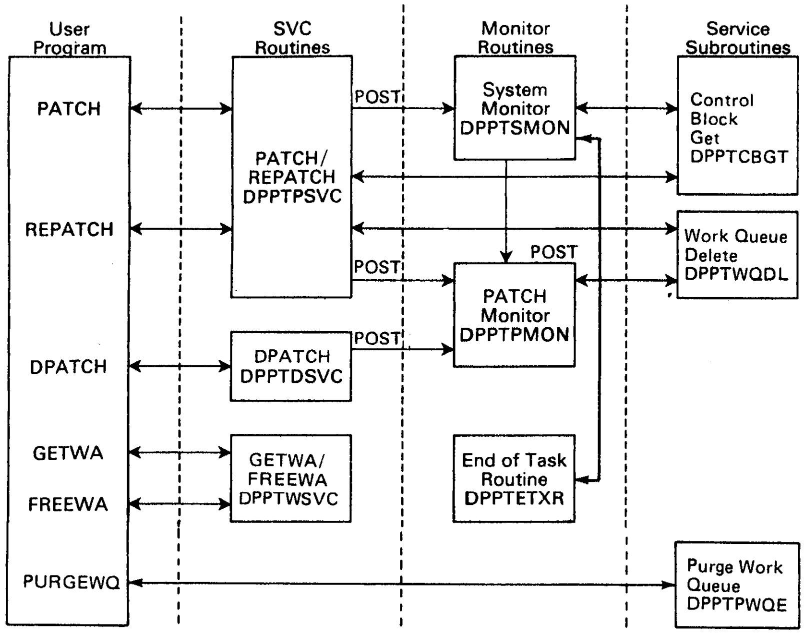


Figure 2-2. Task Management Overview

Task Structure

The Special Real Time Operating System utilizes many tasks (TCBs) during online execution. The task structure for the permanent TCBs is established during initialization. Figure 2-3 shows the Special Real Time Operating System task structure and the task's relative priorities.

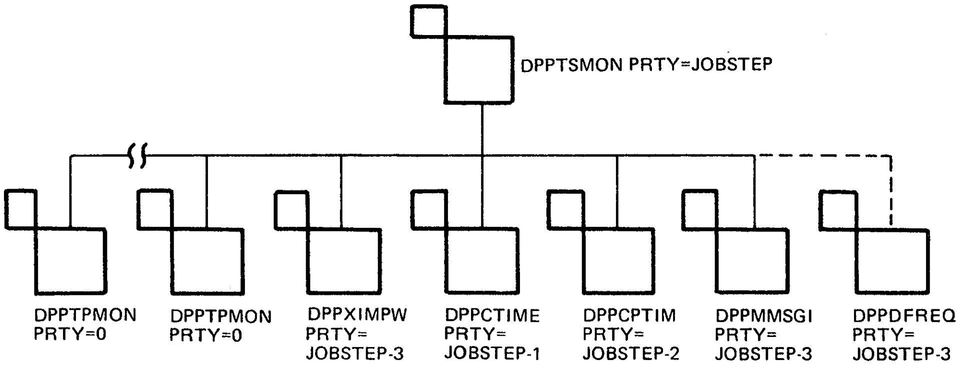


Figure 2-3. The Special Real Time Operating System Task Structure and Priorities

Task DPPTSMON receives control from initialization via XCTL. There will be a variable number of tasks for DPPTPMON. The number will depend upon SYSGEN options. Following initialization these advance TCBs will have a dispatching priority of zero and a limit priority of JOBSTEP task minus three (JOBSTEP-3), which is the highest available user priority.

The task for the input message processor program (DPPXIMWP) is established with a dispatching priority of JOBSTEP-3. Time management (DPPCTIME) has a priority of JOBSTEP-1, and the PTIME monitor (DPPCPTIM) has a priority of JOBSTEP-2. The real-time message handler program (DPPMMSG1) is PATCHed with a priority of JOBSTEP-3.

If cyclic logging (DPPDFREQ) were selected during system generation, the cyclic logging program would be invoked at initialization time and would have a TCB with the priority of JOBSTEP-3. Demand logging does not create a TCB at initialization.

The tasks used by the Special Real Time Operating System are true OS/VS1 tasks and will be assigned OS task priorities based upon the priority of the jobstep task. These tasks compete for resources among themselves and with tasks of other jobs in the system based on their assigned priority. When two or more tasks have the same priority, the order of assignment to that priority value determines which task will be serviced first.

PATCH/REPATCH

PATCH is the service by which a task is created or by which a work request is made for a task already in existence. REPATCH is the means by which a failing PATCH may be retried.

To provide its services, the Special Real Time Operating System builds control blocks and tables which it uses to maintain control of the system and to interface with user programs. Figure 2-4 shows the relationship of the Special Real Time Operating System control blocks for the first PATCH issued on a basic Special Real Time Operating System system, when the user gains control. The Special Real Time Operating System builds its control blocks in protected storage and allocates it via an internal routine called CBGET (control block get). This storage is allocated at initialization and is not expandable.

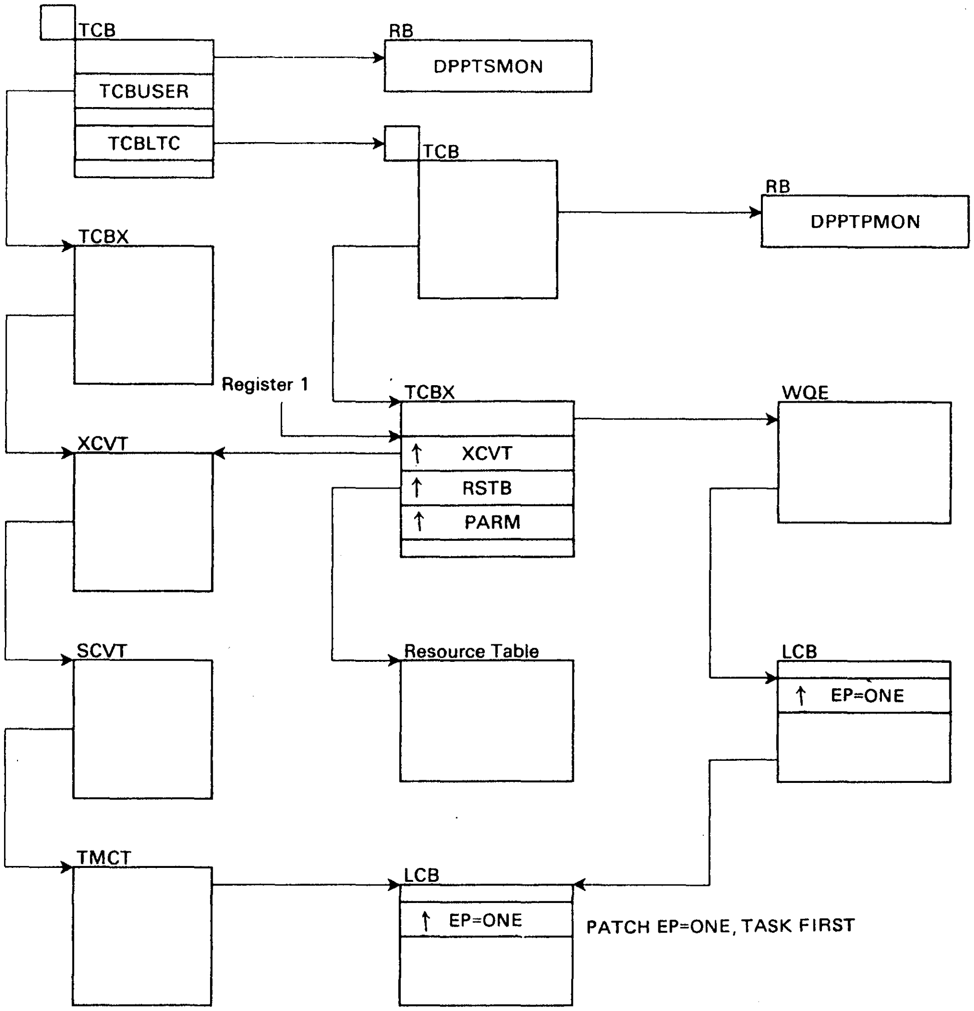


Figure 2-4. Task Management Control Blocks

At the point that program ONE gains control following the PATCH, general register 1 will point to the three words in the TCBX (TCB extension) containing pointers to the XCVT, the resource table, and the parameters being passed into program ONE. The XCVT is the Special Real Time Operating System equivalent to the OS CVT. The XCVT contains pointers and control information which must be available to the subsystems as well as the Special Real Time Operating System. The SCVT contains pointers to system areas and control information which must always be available to the Special Real Time Operating System. The Task Management Control Table (TMCT) contains task-oriented information which must be available to all the Special Real Time Operating System tasks. The TCVX contains control information pertinent only to the specific task and contains a pointer to the XCVT. This pointer links each task to the basic Special Real Time Operating System control information.

The resource table is an 8-byte area of virtual storage that the Special Real Time Operating System gets from subpool zero and passes to the user. This area can be used by the user to pass information across PATCHes to the same task. For example, two PATCHes could be performed for TASK=A, one for EP=A and the second for EP=B. Program A could open a DCB and put its address in the resource table. When program B executes, it could do I/O processing using the open DCB. The resource table is initialized to contain zeros when the task is created and is not changed by the Special Real Time Operating System as long as the task is in existence.

The work queue element (WQE) is built by the Special Real Time Operating System to represent the PATCH request for execution of program ONE under task FIRST. Once program ONE has completed execution and returned control to the Special Real Time Operating System, the WQE is deleted. If additional PATCHes had been made for task FIRST, additional WQEs are queued to the TCBX. When the first execution of program ONE is completed, the first WQE is removed and the second scheduled. This process continues until there are no WQEs left on the queue. There is one WQE created for every PATCH.

The load control block (LCB) is created by the Special Real Time Operating System to represent the load module for program ONE. Program ONE in Figure 2-4 is represented by two LCBs. This is the case when the program is reentrant. A non-reentrant module is represented by only one LCB chained to the requesting WQE. There is one LCB for each module (EP=) under each task (TASK=), plus one LCB for each reentrant module in the partition. LCBs are created for modules loaded through the use of the Special Real Time Operating System services. Figure 6 shows the Special Real Time Operating System LCB-WQE blocks that will be built for the following PATCHes:

EXAMPLE 1:

•	PATCH 1	PATCH	TASK=X,EP=A	(reentrant)
•	PATCH 2	PATCH	TASK=X,EP=B	(non-reentrant)
•	PATCH 3	PATCH	TASK=Y,EP=A	(reentrant)
•	PATCH 4	PATCH	TASK=X,EP=B	(non-reentrant)
•	PATCH 5	PATCH	TASK=Y,EP=B	(non-reentrant)
•	PATCH 6	PATCH	TASK=Y,EP=A	(reentrant)

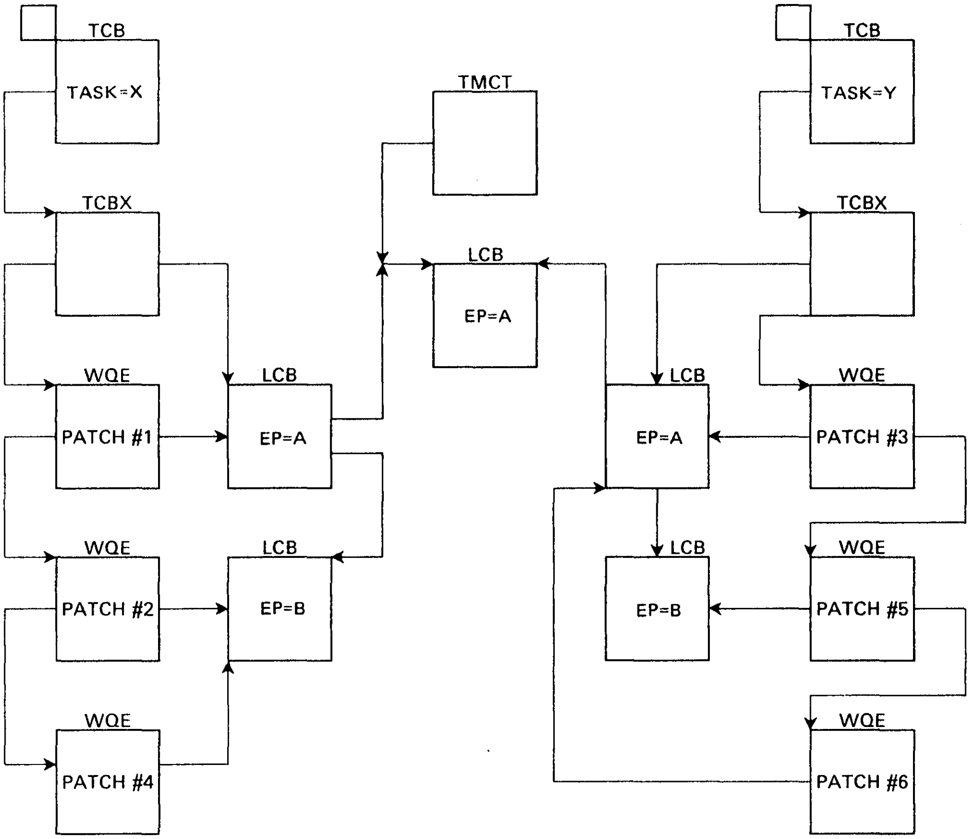


Figure 2-5. Control Blocks Built for Example 1

The control blocks will be built as shown in Figure 2-5 at a point in time when all PATCHes have been issued, and program A and B are executing, under the first WQE.

PATCH 1 causes a WQE to be scheduled for program A on task X. PATCH 1 also creates the LCB pointed to by the WQE and the LCB pointed to by the TMCT.

PATCH 2 creates the WQE and its LCB for task X, program B.

PATCH 3 creates the WQE on task Y and the LCB for program A pointed to by the WQE. It also points the LCB to the existing LCB for program A on the TMCT.

PATCH 4 creates the WQE for task X and points it to the LCB for program B that was previously created by PATCH 2.

PATCH 5 creates the WQE for task Y, and because this is the first request for program B on task Y, creates an LCB for B and chains it to the WQE.

PATCH 6 creates a WQE for task Y and points it to the LCB previously created by PATCH 3.

The Special Real Time Operating System task management is initialized by DPPINIT. DPPINIT gets protected core from subpool 253 and builds the XCVT, the SCVT, and the TMCT. It then initializes the get work area (GETWA) and control block get (CBGET) storage. Next, initialization determines the number of TCBs and task control block extensions (TCBXs) to be obtained and initialized, and creates the TCBs by attaching the PATCH monitor (DPPTPMON) for the number of TCBs. Next, DPPINIT gets CBGET storage for TCBXs, chains the TCBX to a TCB, and puts the TCBX on the TMCTFREE chain. When initialization is completed, it XCTLs to the system monitor (DPPTSMON). At this point the system is configured as shown in Figure 2-6.

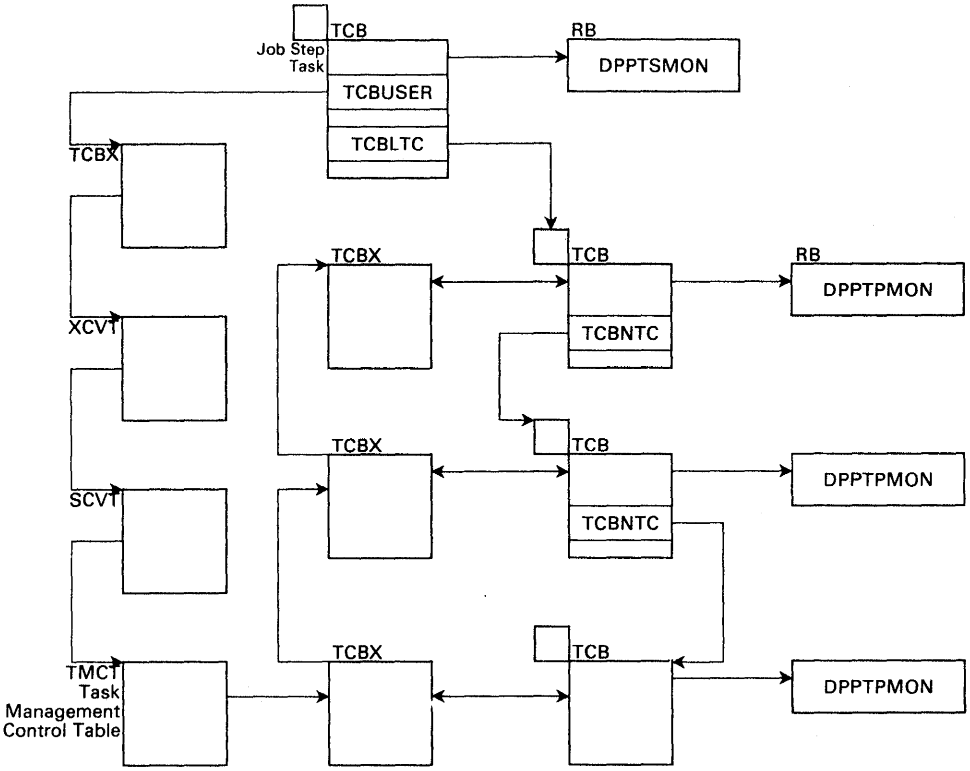


Figure 2-6. Control Blocks After Initialization

When DPPTPMON is in storage and begins execution, it waits until it is posted by DPPINIT. DPPINIT posts DPPTPMON when a TCBX has been initialized and chained to the TCB. DPPTPMON then does a GETMAIN for the resource table, and chains it to the TCBX. A STAE macro call is then executed by DPPTPMON specifying load module DPPTSTAE as the STAE exit routine. DPPTPMON then executes a WAIT macro call. At this point the Special Real Time Operating System is ready for user service requests.

The user requests that a task be brought into virtual storage and executed via the PATCH macro. Two different types of tasks can be executed. Dependent tasks operate similarly to normal OS/VS1 tasks; the requested module is loaded and executed once; then the module is deleted, and the task is terminated. Independent tasks, however, can request loading of multiple programs; each can be executed many times and is terminated only upon a specific request from a user by the DPATCH macro. To facilitate multiple execution of independent tasks, the Special Real Time Operating System loads each reentrant program only for the initial PATCH. The WQE and LCB are built and queued to the tasks TCBX. On subsequent PATCHes to the same task requesting the execution of the same program (EP=), a WQE will be created; but the program will not be reloaded, and the WQE will be pointed to the LCB created by the first PATCH.

An option on the PATCH macro allows programs to be deleted after the WQE has been processed, even though the program is reentrant.

The user's PATCH macro results in the PATCH SVC code (DPPTPSVC) gaining control. The SVC validity checks the input parameters, and if they are valid, obtains a TCBX and a TCB for the task. DPPTPSVC then builds a WQE and an LCB for the program and chains them to the TCBX. DPPTPSVC then posts DPPTSMON to change priority (CHAP) of the TCB to the specified priority (PRTY=) and returns control to the program that issued the PATCH SVC.

The system monitor (DPPTSMON) has three main functions.

- When posted by DPPTPSVC, it CHAPs the TCB to the requested priority.
- It creates TCBs and TCBXs and maintains them on the TMCT chain.
- DPPTSMON handles the loading and deleteing of reentrant modules.

The PATCH monitor (DPPTPMON) is the Special Real Time Operating System-user interface, DPPTPMON manipulates WQEs and non-reentrant LCBs. DPPTPMON is the program under which all user tasks are executed. When the program has been loaded and the WQE scheduled, DPPTPMON branches to the user code. Upon completion, the user executes a normal BR14 return, and DPPTPMON regains control, posts the user ECB, and attempts to schedule the next WQE. If no WQEs exist, DPPTPMON waits for the next PATCH.

Note: Because user programs are loaded and branched to by DPPTPMON, the user program will not be represented by an RB on the OS/VS1 system. As a result, user programming errors will cause ABEND dumps that show DPPTPMON as the ABENDing program.

The following examples show how the user would invoke the Special Real Time Operating System task management services through the use of the PATCH macro.

```
PTCH01      PATCH   TASK=ONE,EP=FIRST,QL=3,         *
                    QPOS=FIRST,PRTY=(TWO,4),        *
                    ECB=(ECBONE),                   *
                    FREE=P,ID=4
```

This PATCH will cause a task with the name ONE to be created. If task ONE already exists, a WQE will be queued to it to represent PATCH PTCH01. PTCH01 is a request for execution of program FIRST, and the WQE will be queued at the top of the queue (QPOS=FIRST). If the task does not exist, it will be created with a priority of 4 less than the existing task named TWO and will allow a maximum of three WQEs (QL=3) to be queued plus the current WQE being processed. If task TWO does not exist, the PATCH will not be processed, and the PATCHor will be given a return code 10. If task ONE does exist, the QL and PRTY keywords will have no effect. The ECB=keyword specifies that an ECB at location ECBONE is to be posted when the Special Real Time Operating System task management dequeues the WQE which represents this PATCH request. The ECB will be posted with a Special Real Time Operating System task management POST code in the high-order byte and the low-order three bytes of register 15 if the PATCHed program is completed successfully, or the ABEND code is in the low-order three bytes if the task ABENDed. FREE=P requests that the Special Real Time Operating System task management services free (FREEMAIN) the virtual storage occupied by the PROBL (user problem parameter list). The ID=4 requests that a value of 4 be put into the PROBL and passed to the PATCHed program.

```
PTCH02      PATCH    ID=255,TASK=MAIN                 *
                     EP=WORK01,PRTY=(,1),             *
                     QL=9
```

PTCH02 uses the special ID (ID=255). This ID creates a Special Real Time Operating System task named MAIN with a queue length of 9 and a priority of 1 less than the PATCHor. The program named WORK01 is loaded but never given control, because the ID is 255. This facility allows the user to create a task structure of reentrant and serially reusable programs. As a result, he knows the task structure prior to the execution of the PATCHed tasks.

Reentrant and serially reusable programs are kept in virtual storage and are not deleted at completion of their execution. If the user wishes to have a reentrant or serially reusable program deleted at its completion, he must code the PATCH with EP=(name, DELETE). This will result in the LCB for the program being removed from the task's LCB chain, and if no other tasks have issued PATCHes for the program, the load module will be deleted. However, if other tasks did PATCH the program and did not request the DELETE option, the load module will not be deleted. If multiple tasks PATCH a module and all specify the DELETE option, a use count is kept by the Special Real Time Operating System task management, and the module is deleted when the use count becomes zero. The Special Real Time Operating System use count is independent of OS/VS1 use count. As a result, if a user program does a LOAD, followed by PATCH with EP=(name,DELETE), the Special Real Time Operating System DELETE will not necessarily result in the module being removed from virtual storage, as the OS/VS1 use count will not go to zero. This is because the Special Real Time Operating System task management routines will issue LOAD for the module on the first PATCH to it resulting in an OS/VS1 use count of 2.

WORK Queue Pooling

Work queue pooling is a capability of Special Real Time Operating System to allow a single task to process work that would otherwise be processed by several tasks, or several tasks to process the work that would otherwise be processed by a singel task, or combinations thereof. A close similarity to this concept can be observed in the OS/VS1 job scheduler where an initiator can process work from several job classes or jobs of a given class can be processed by any of several initiators.

Work queue pooling may be invoked for a given execution of the Special Real Time Operating System by including in the initialization stream the commands which define the elements, Queue Holders (QH) and Queue Processors (QP), to be active on this execution. To make use of work queue pooling, the user will execute PATCHes to the queue holders, exactly as done to independent task. One command (card) will define one QH or QP. The QP represents a Special Real Time Operating System and OS task, the same as with an independent task. It is defined at initialization and will remain for the duration of the job. There is no provision for adding or deleting QPs after initialization. The QP differs from an independent task in that work cannot be passed directly to the QP via a PATCH.

The QH appears as an independent task without an associated OS task. Work is passed to the QH via PATCH but the work is processed by one of the QPs associated with the QH. The QH has a name, exactly as an independent task and the TASK= operand of the PATCH and other macros will reference the QH by this name. As with QPs, all QHs must be specified at initialization. When specifying QHs, the user assigns the name and other attributes. Any QH may be specified to be connected to several QPs; that is, any of the connected QPs are allowed to process work that is 'PATCHed' to this QH. Also, several QHs may be connected to any one QP, which means that a QP can process work from any of several QHs. There is an implied priority relationship in this scheme in that when a QP completes a piece of work, it will look for work in the first QH connected to it and only if that QH is empty will it look to the next QH, etc. The opposite is also true when a piece of work is passed to a QH, the work will be given to the first QP that is connected to it and is not busy. If all connected QPs are busy, the work will be queued to the QH to await a QP that becomes available.

The relationships between QPs and QHs is defined through the initialization stream commands. The QP command allows the user to specify the order in which the QP is to search the QHs for new work when a piece of work is completed. The ordering of the QP commands implies the order in which to search for an available QP when work is added to a QH. Each QP is assigned a number (0 to 99) on the OP command. From this number a name is generated. The user assigned number will be used for all references by other commands. Each QH is assigned a name (1 to 8 EBCDIC characters) by the QH command. This name will be used for all references to it, either by other commands or by programs via the PATCH macro, etc. Various other parameters may be specified on the QH and QP commands. The PRTY= parameter on the QP command is similar to the same parameter on the PATCH command. The HOLD=YES parameter allows the QP to be initialized in a hold status which meands that it will not process any work until a release is entered through the IMP commands provided.

The QL= parameter on the QH command specifies the number of work queues that can be stacked for this QH, similar to the same parameter on the PATCH command. The parameter SEQ=YES specifies that only one QP may be processing work from this QH at any time. The HOLD=YES parameter specifies that no work is to be processed from this QH. The PATCH=NO parameter specifies that the PATCH processor is to reject all PATCHes to this QH. The SEQ=, HOLD= and PATCH= parameters can be modified during execution through the IMP command processing provided.

The inclusion of work queue pooling on a given execution of Special Real Time Operating System does not effect independent or dependent task operations. When a PATCH is executed, the PATCH code will search for a TCBX with the task name equal to that specified on the PATCH. If the name is not found or a name is not specified (dependent task) a Special Real Time Operating System task is created and the work queued to the new task. If the name is found, the work is added to the work queue of the TCBS. If the TCBX is a QH, the work participates in the queue pooling.

The user of Work Queue Pooling has the ability to determine the status of and control certain functions of the QPs and QHs through the IMP command processor. The user can hold or release either a QP or QH. If a QP is held, it will not accept any new work. If a QH is held, the QP(s) will not take work from it. The user can set a QH to be sequential or non-sequential. In the sequential state, only one QP may be processing work from this QH at any time. Non-sequential is the normal state where all connected QPs may be processing work from this QH simultaneously. The QH can be set to a PATCH or NOPATCH state. In the NOPATCH state all PATCHes to it will be rejected. PATCH is the normal state. In addition to changing one of the above conditions, the command can cause all work to be specified QH to be purged.

The IMP command can cause Special Real Time Operating System messages to be output to report the status of these states as well as other information about the QPs or QHs. This information will include the element (QP and QH) name, the names of the elements connected to it, and the number of work queue elements awaiting processing.

When a program receives control as the result of a PATCH, Register 1 contains the address of a 3 word table. The second word of this table contains the address of a resource table. If the program is executing under control of a QP, this resource table is associated with the QP. Every program that is PATCHed to execute under a given QP will receive this same resource table. In addition, register 0 will also contain the address of a resource table. If the program is executing under control of a QP, this resource table will be associated with the QH from which the work was taken. This means that all programs which execute as the result of a PATCH to a given QH will have access to the same QH resource table. Caution must be exercised by the user if the QH is connected to two or more QPs, since several programs may be competing for this resource table. If the program is executing under Special Real Time Operating System task (not a QP) register 0 will contain the same address as is in the second word of the table addressed by register 1.

The following example shows how QP and QH statements in the SYSINIT input stream can be used to define two queue processors and two queue holders. All other control statements in the input stream have been omitted.

```
//SYSINIT      DD

            QP       19,QH=(DPPQABC,DPADMNO)
            QP        2,QH=(DPADMNO,DPPQABC)
            QH       DPPQABC
            QH       DPADMNO
```

In this example both queue processor 19 (QP19) and queue processor 2 (QP02) have been created to process work from queue holders DPPQABC and DPADMNO. However, since in the QP statement for QP19, queue holder DPPQABC has defined first, QP19 will give it a higher logical priority. Since the inverse is true in the other QP statement, QP02 will process the work from DPADMNO before processing work from DPPQABC.

Assume the following PATCH macro calls are executed to route work to the queue holders.

```
        A   PATCH   TASK=DPPQABC,...
        B   PATCH   TASK=DPADMNO,...
        C   PATCH   TASK=DPPQABC,...
        D   PATCH   TASK=DPPQABC,...
        E   PATCH   TASK=DPPQABC,...
```

The resulting task/queue structure is illustrated in Figure 2-6.1.

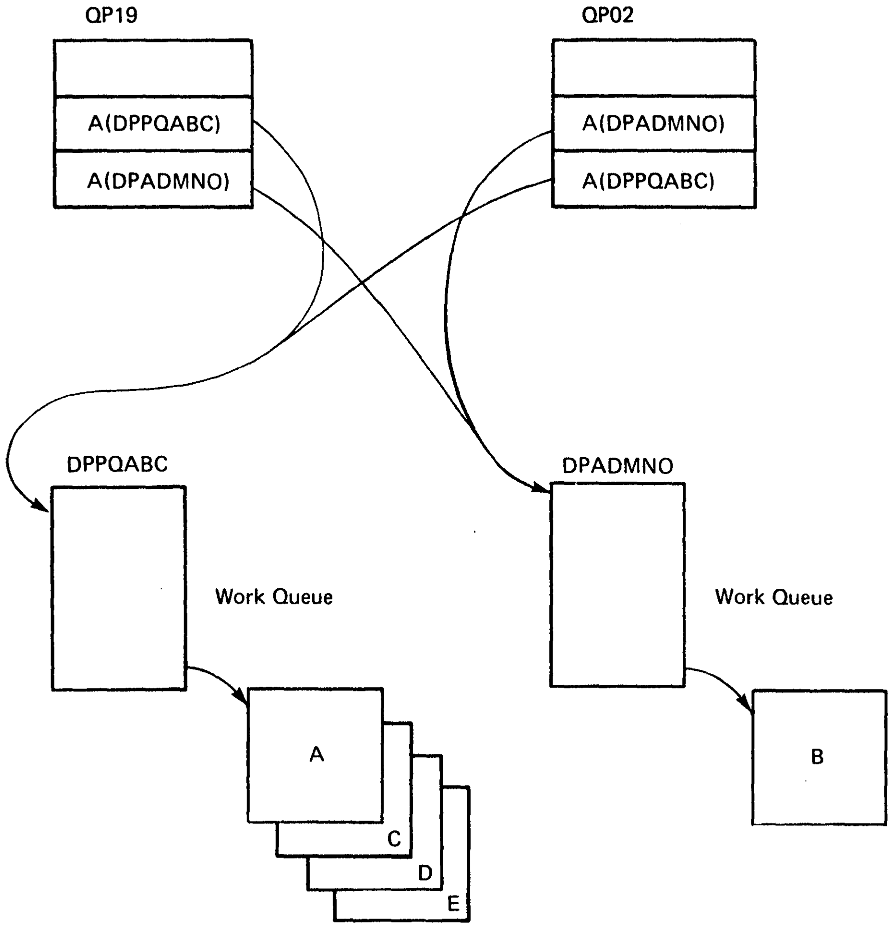


Figure 2-6.1. Task/Queue Structure

QP19 will select work queue A from queue holder DPPQABC and QP02 will select work queue B from queue holder DPADMNO. Assume QP19 completes work queue A before QP02 completes work queue B. When QP02 completes work queue B, QP02 will attempt to select additional work from queue holder DPADMNO and, finding it empty, will select work queue D from queue holder DPPQABC. Upon completion of work queue D, QP02 will again attempt to select work from queue holder DPADMNO and, finding it still empty, will select work queue E from queue holder DPPQABC.

Using a similar example to illustrate the functions of some of the optional parameters on the QP and QH statemtns, assume the following SYSINIT input stream was specified. Again only the QP and QH statements will be shown.

```
//SYSINIT       DD
                QP       19,QH=(DPPQABC,DPADMNO,DPPQXYZ)
                QP        2,QH=(DPPQXYZ,DPPQABC),PRTY=(JOBSTEP-0)
                QH       DPPQXYZ,SEQ=YES,QL=10
                QH       DPPQABC
                QH       DPADMNO,HOLD=YES
```

In the second example, queue processor number 19 (QP19) has been created with a default dispatching priority of the job step task minus 8 to process work queued in queue holders DPPQABC, DPADMNO, and DPPQXYZ and queue processor number 2 (QP02) has been created with a dispatching priority of the job step task minus 3 (the highest allowed to any user task) to process work queued in queue holders DPPZXYZ and DPPQABC (see Figure 2-6.2).

Queue holder DPPQXYZ has been created as a sequential queue holder with a queue length of 10. Queue holders DPPQABC and DPADMNO have been created with default queue lengths of 255 (see Figure 2-6.2). DPADMNO has been held, that is PATCHes specifying a task name of DPADMNO will be accepted but neither queue processor (QP19 or QP02) will be permitted to select work from that queue holder.

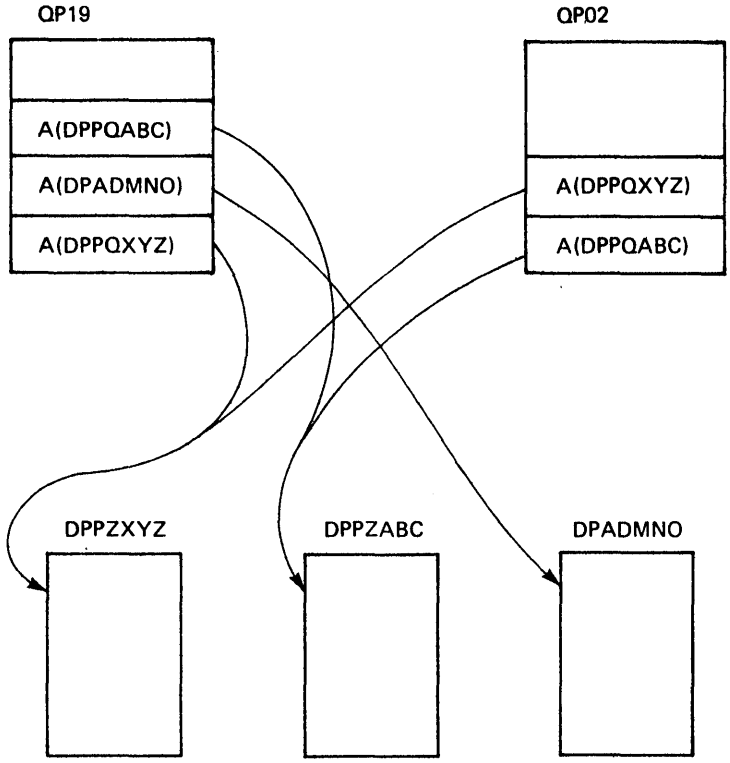


Figure 2-6.2. Queue Processor/Queue Holder Structure

Now assume the following PATCH macro calls are executed to route work to the three queue holders.

```
        A    PATCH    TASK=DPPQXYZ,...
        B    PATCH    TASK=DPADMNO,...
        C    PATCH    TASK=DPPQABC,...
        D    PATCH    TASK=DPPQXYZ,...
        E    PATCH    TASK=DPPQXYZ,...
```

The resulting task/queue structure is shown in Figure 2-6.3.

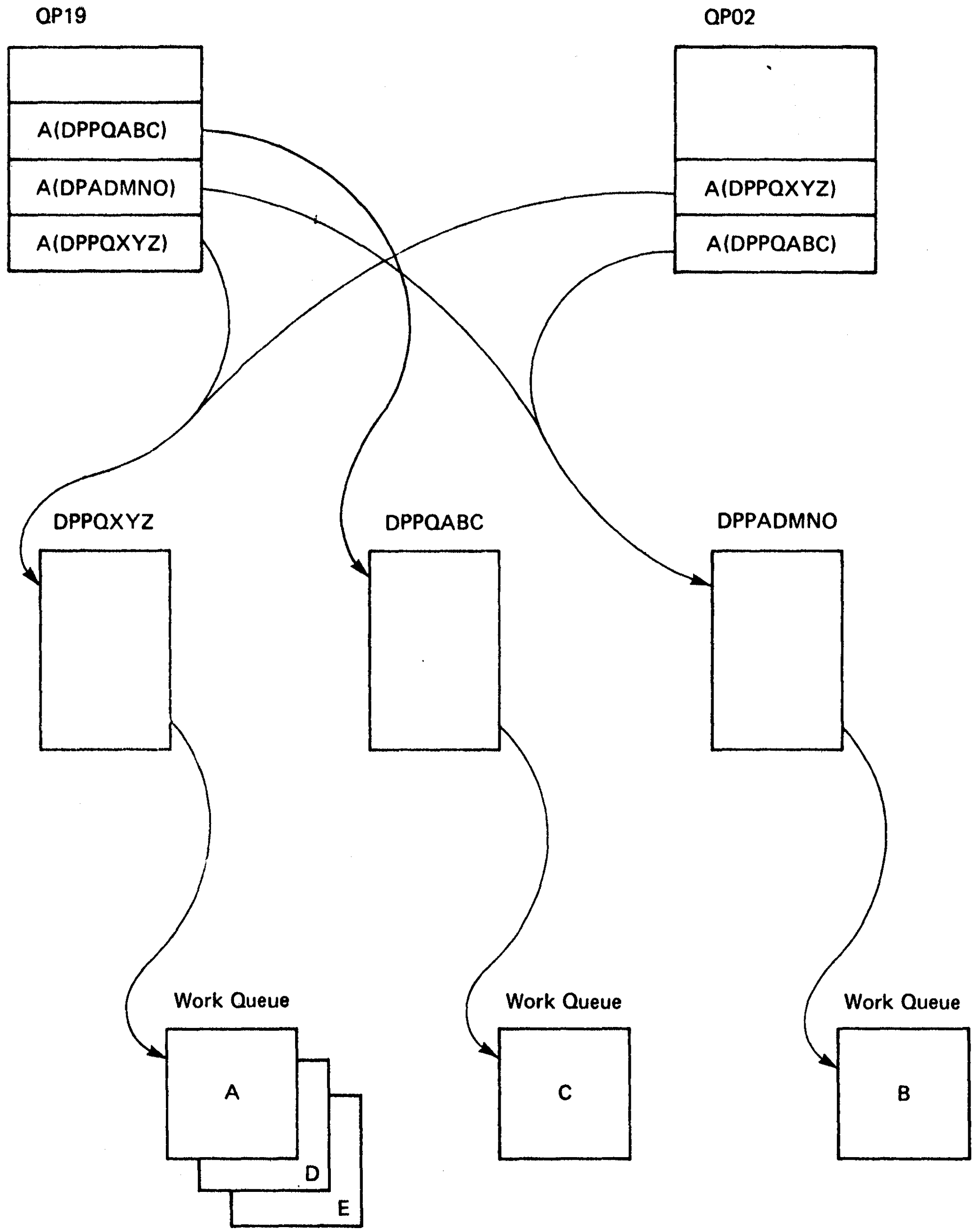


Figure 2-6.3. Task/Queue Structure

QP02, having the higher priority, will select work queue A from queue holder DPPQXYZ. QP19 will select work queue C from queue holder DPPQABC.

Assume QP19 completes work queue C before QP02 completes work queue A. QP19 will try to select additional work from queue holder DPPQABC first; (see Figure 2-6.4) but since all work for that queue holder has been exhausted, QP19 will then try to select work from queue holder DPADMNO. However, since DPADMNO was defined on the QH statement as being held, the work queue B cannot be selected by any queue processor. Therefore, QP19 will then attempt to select work from queue holder DPPQXYZ. Since queue holder DPPQXYZ was defined on the QH statement as being sequential and queue processor QP02 is currently executing a work queue from DPPQXYZ (work queue A), QP19 will be unable to select work from this queue holder either. Having searched for work on all queue holders that QP19 can process and having found none, QP19 will then be placed in a wait state.

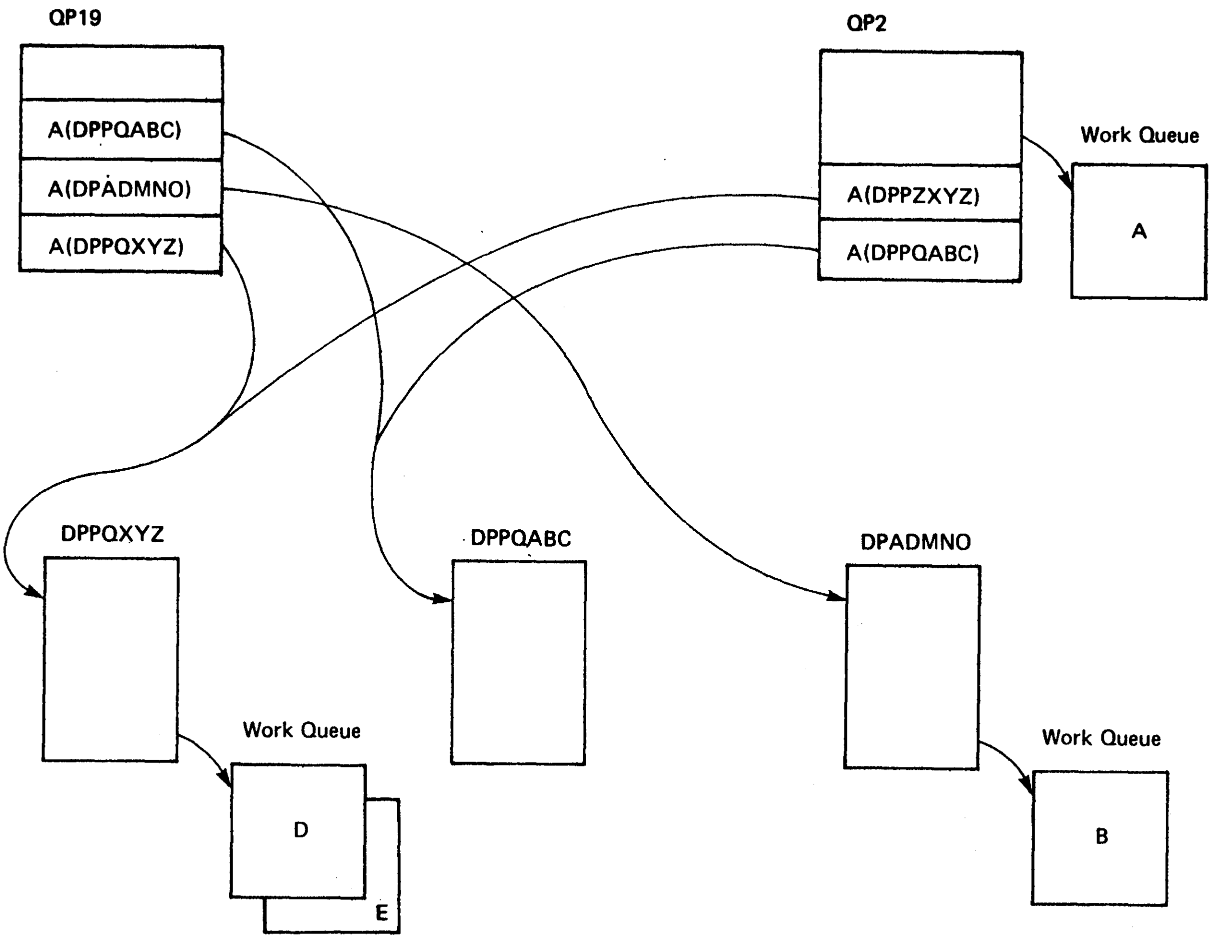


Figure 2-6.4. Task/Queue Processing

If a QS command of the form r xx,QS,AALQH,REL were to be issued, then QP19 would then be allowed to select work queue B from queue holder DPADMNO.

DPATCH

The DPATCH macro is used to stop the processing of a specified task and to cause the program to be deleted. Since the task may have several entries on its work queue, four types of DPATCHes allow flexibility.

First, the TYPE=I causes the task to be DPATCHed immediately. The task is not allowed to complete the processing of the current WQE, but is ABTERMed. If ECB= was specified at PATCH time, the ECB is posted with the ABEND completion code hex'4C'. The ECBs for further WQEs are posted with a DPATCH completion (hex'42).

Second, the TYPE=U causes the task to be DPATCHed when the current WQE completes, and the ECBs for remaining WQEs are posted with a DPATCH completion.

Third, TYPE=C causes the task to be DPATCHed only if there are no WQEs when the DPATCH request is received.

Fourth, TYPE=W causes the task to be DPATCHed only when the work queue becomes empty. Additional WQEs can be added after the DPATCH request, and the DPATCH would only occur after the queue becomes empty.

Fifth, TYPE=A causes the program being executed under the specified task to be ABENDed without deleting the task or any WQE's that may be awaiting execution.

Note: If QPOS=DPATCH was specified on any one of the PATCHes to a given task, that WQE is scheduled, and the program executed at DPATCH time before the task is removed from the system.

PURGEWQ

The Purge Work Queue Facility provides the capability of selectively purging work requests to a specified independent task. The selected work requests will be removed from the active work queue (i.e., a chain of work requests that have been generated in response to PATCH macro calls but have not yet been executed) or from the DPATCH work queue (i.e., a work request generated in response to a PATCH QPOS=DPATCH,...macro call). Other work requests for that task will not be purged but will be allowed to execute normally.

PURGEWQ, on request, also notifies the user whenever the last of the selected work requests has been purged.

The current work request (i.e., work request currently in execution for the specified task) will not be purged but will be allowed to complete normally eventhough it may be one of the selected work requests. PURGEWQ, in this case, will notify the user after the specified task has completed the execution of the selected work request. In addition to providing the synchronization of the completion (or purging) of selected work requests, PURGEWQ can be used in a "work shedding" environment as well. For example, work requests deemed to be of lesser importance can be selectively purged from the queue of work requests for a specified independent task to allow more time for the more important work requests to execute. The execution of a PURGEWQ macro call will not prohibit the scheduling of future work requests (PATCHes) to the specified independent task. PURGEWQ operates only on those work requests that have previously been scheduled.

End of Task Exit Routine

The Special Real Time Operating System end of task exit routine (ETXR) is the program (DPPTETXR) that gains control from OS/VS1 upon termination of a Special Real Time Operating System task. The ETXR routine executes under the jobstep task (DPPTSMON) and cleans up after task termination.

In the event that a task ABENDs, DPPTETXR issues a message through the real-time message handler specifying the task name and the failing program EP name. The TCBX is saved, and the TCB is detached. DPPTETXR also posts DPPTSMON to have the TCBX chained to a new TCB.

If the task is terminating normally, the TCB is detached. In either case, normal or abnormal termination, control of all locked resources is released and GETWA type AT areas are freed.

STAE Processing

Two STAE exit routines are used (1) to provide an interface to a user exit routine and to provide the Special Real Time Operating System with a DUMP/NODUMP facility upon abnormal termination of a subtask and (2) to allow cleanup functions to be performed when the real-time job step task is terminating.

An initialization input stream command, STAEX, allows the user to specify the name of the user coded load module (exit routine) which is to be given control when any one of a list of load modules encounters an ABEND. A STAE is invoked for every Special Real Time Operating System task so that when an ABEND occurs in one of the so specified load modules while executing under a Special Real Time Operating System task, including QPs, the exit routine will be given control before the DUMP/NODUMP decision is made by the standard Special Real Time Operating System STAE processor. Within the exit routine, the user may schedule a retry routine, force the ABEND to proceed with a dump or allow the Special Real Time Operating System STAE option in effect to determine if a dump is to be taken.

On entry to the user exit routine, registers 0, 1, 13, 14 and 15 will contain the values as defined by OS/VS1 STAE interface routines (see *OS/VS1 Planning and User Guide*, STAE macro instruction). Register 2 will contain the address of the TCBX for the abending task. In a queue pooling environment this will be the address of the QP TCBX. The user exit routine is limited by the same restrictions as a normal STAE exit routine.

The DUMP/NODUMP facility allows control of System ABEND dumps for all load modules, for a group of load modules, or for an individual load module. This facility will not suppress user ABEND dumps. It is invoked by an entry to the Input Message Processor (IMP) of the form:

```
                   ┌              ┐
                   │⎧,DUMP       ⎫│
r xx,STAE[,SLAVE]  │⎪,NODUMP     ⎪│  [modulename,modulename,...]
                   │⎨,ONEDUMP    ⎬│
                   │⎪,STEP       ⎪│
                   │⎩,OPTION     ⎭│
                   └              ┘
```

The first positional operand, STAE, is required and defines the reply to the Input Message Processor as a command to the DUMP/NODUMP service interface routine.

The second operand, SLAVE, is used by the Input Message Processor to route the command to the DUMP/NODUMP service routine in the SLAVE partition only. It is not a positional operand in that a null field (double comma) is not required to denote its absence.

Examples:

```
    r xx,'STAE,NODUMP'
```

This IMP command will cause all system ABEND dumps to be suppressed for the MASTER partition.

```
    r xx,'STAE,SLAVE,NODUMP'
```

This IMP command will cause all system ABEND dumps to be suppressed for the SLAVE partition.

The third operand is used by the DUMP/NODUMP service routine in establishing the options that will be in effect for those modules This operand is a positional operand, and its absence must be denoted by a null field (double comma). If omitted, the DUMP option will be assumed for those modules specified in this reply.

The valid options are:

DUMP - allows a dump to be taken for these modules (provided there is a SYSUDUMP or SYSABEND DD statement).

NODUMP - suppresses a dump from being taken for these modules.

ONEDUMP - allows one dump to be taken for these modules (provided there is a SYSUDUMP or SYSABEND DD statement) then suppresses any more dumps for that module.

STEP - ABENDs the job step if one of these modules ABENDs.

OPTION - allows the operator to choose whether or not to take a dump following an ABEND of these modules. The operator is informed of the ABEND via a WTQR (message 850) and must reply 'YES' to receive the dump.

The remaining operands, if any, are used to indicate the load module(s) that are to be covered by the specified option. A maximum of 10 load module names may be specified on any one reply. Null fields (double commas) will not be accepted.

Example:

```
    r xx,'STAE,NODUMP'
    r xx,'STAE,ONEDUMP,MODA,MODB'
```

These two IMP commands will cause all system ABEND dumps to be suppressed for the MASTER partition except ABENDs for modules MODA and MODB. One dump will be taken for MODA on ABEND, and one dump will be taken for module MODB on ABEND after the command is entered.

The use of a question mark (?) to terminate a load module name indicates that the specified option is in effect for all modules beginning with the portion of the name specified. The portion of the name specified must be at least one character and must not exceed seven characters. The modules are processed as a group and not as individual modules. This means that if the ONEDUMP option is specified with the module name DPP?, only one dump would be taken for the first module to ABEND with a name beginning with the three characters DPP. Dumps will be suppressed for any subsequent ABENDs for modules which have names beginning with DPP.

Example:

```
    r xx,'STAE,NODUMP'
```

```
r xx,'STAE,STEP,MODUL?
```

These two IMP commands will cause all system ABEND dumps to be suppressed for the MASTER partition except that a system ABEND from any module with a name beginning with MODUL will dump and will ABEND the entire jobstep.

If no load module name is provided on a reply, the specified option will be in effect for all load modules regardless of any previous DUMP/NODUMP service command. This will allow the option to pe reset without having to cancel each previous command. Providing one or more load module names will set (or reset) the option for only those modules specified on that command. Any previous DUMP/NODUMP service commands for other modules will not be modified and will remain in effect.

Note: The options in effect at the time of the ABEND are the options that will be honored except that dumps for step ABENDs are not suppressed. It should also be noted that the user exit routine invoked in response to the STAEX statement in the SYSINIT input stream will receive control before the STAE option processing is initiated. Any request by that routine to retry or bypass STAE option processing will take precedence over the STAE IMP command option in effect.

Upon abnormal termination of a subtask executing under the real-time job step task, one of the Special Real Time Operating System STAE exit routines (DPPSTAE) will gain control. This routine will then examine the STAE command options in effect at the time of the ABEND to determine whether or not a dump should be taken for this task.

Upon termination of the real-time job step task, another Special Real Time Operating System STAE exit routine (DPPISTAE) will gain control. This routine will unfix any storage previously fixed by the DPPIPFIX routine and clear the external interrupt handler flags. If the job step terminating is a MASTER job, the corresponding SLAVE job is also terminated (USER ABEND code of 41). If the job step terminating is a SLAVE job, the corresponding MASTER job is located, and the MASTER job's two-partition flags are turned off.

It is important to note that there are certain conditions in which the STAE routine is not given control when the real-time job step is terminated (e.g., an operator CANCEL command) and these cleanup functions cannot be executed. Therefore, the user must use care and terminate a real-time job step by a reply to the Input Message Processor of the form

```
r xx,CANCEL[,...]
```

If the SLAVE partition has terminated with a supervisor ABEND code of 122, 13E, 222, 322, or 722, an IMP command of the form

```
r xx,CANCEL,SLAVE
```

will ensure that the two-partition flags in the MASTER partition will be reset even though the SLAVE partition job is no longer active.

Dynamic Load Module Purge

The Dynamic Load Module Purge Facility permits the system operator to cause a load module, which has been loaded in response to one or more PATCH requests, to be deleted from virtual memory. Thus, the user can redefine a load module in the library (JOBLIB, STEPLIB, or LINKLIB) and purge the in-memory copy, so that when the load module is next requested, the new copy will be fetched. The redefinition may entail replacing the existing copy of the load module or adding a copy on a data set that is searched ahead of the one on which it was originally found. The redefinition can be done in a background partition or in a backup System/370 which shares disks with the online System/370. Through the use of this facility, the new load module can be integrated into the online system without otherwise disturbing the job.

This procedure is not necessary for modules that are link-edited as non-reentrant because they are fetched from the library for each execution. Those modules that are represented in a system BLDL list are not normally affected by this procedure since the disk address of those modules is resolved at system IPL time and cannot be re-resolved except by re-IPL. Those modules that are identified in a Resident Access Method (RAM) list are loaded at IPL time and as such are also not affected. This procedure affects only those modules that are invoked through a PATCH or PTIME service and not those which may be loaded, attached, linked, or XCTLed to outside of the PATCH interface.

Dynamic Load Module Purge is invoked by a reply to the Special Real Time Operating System Input Message Processor.

```
   r xx,DLMP,[SLAVE,]time,modulename,modulename...
```

DLMP defines the reply as a command to purge the modules specified. Up to 10 module names may be specified with one command. If SLAVE is specified, the purge operation is performed in the SLAVE partition; if it is omitted, it is performed in the MASTER (or only) partition. A time value may be specified on the command as a decimal integer between 0 and 1200; if omitted, a default of 2 is used. This value defines the maximum number of seconds that the DLMP program will wait to allow other tasks to complete execution of the specified load modules. Therefore, this value plus the necessary time for all DELETE operations is also the time that all other tasks with a request for one of the specified modules may have to wait before they are permitted to use their module.

In response to the request, the Dynamic Load Module Purge program DPPTDLMP searches the TCB extensions for the Special Real Time Operating System tasks that have requests for, or are currently using, the specified load module(s). If the task is not currently using one of the modules, it will not be permitted to resume using it until the purge operation has been completed. If a task is currently using the load module, a flag is set, and the current use is permitted to complete, but the task cannot process another WQE that requests a module in purge until the purge operation is completed.

However, only those tasks are quiesced that have a WQE on top of the queue which requests a module that is in purge; every other execution continues undisturbed. DPPTDLMP waits the specified time for the using tasks to complete execution of the modules. If the time expires before all tasks are through, the operation is abandoned, and messages DPP021 will specify the name of those modules that were not completed, plus message DPP022 will specify that the operation is abandoned. If all tasks complete using one or more of the specified modules in time, DPPTDLMP causes the module(s) to be deleted and message DPP023 will specify that the operation is completed. In either case, all tasks that had been quiesced are then allowed to resume normal operation.

TIME MANAGEMENT

The Special Real Time Operating System provides time management facilities to meet the requirements of a real-time operating system. The Special Real Time Operating System time management services fall into two major categories. First, the Special Real Time Operating

System time and date are maintained independently of the OS/VS1 time and date. Second, the capability of issuing PATCHes on a cyclic time interval is provided through the PTIME macro call. Figure 2-7 is a block diagram of PTIME logic and control flow.

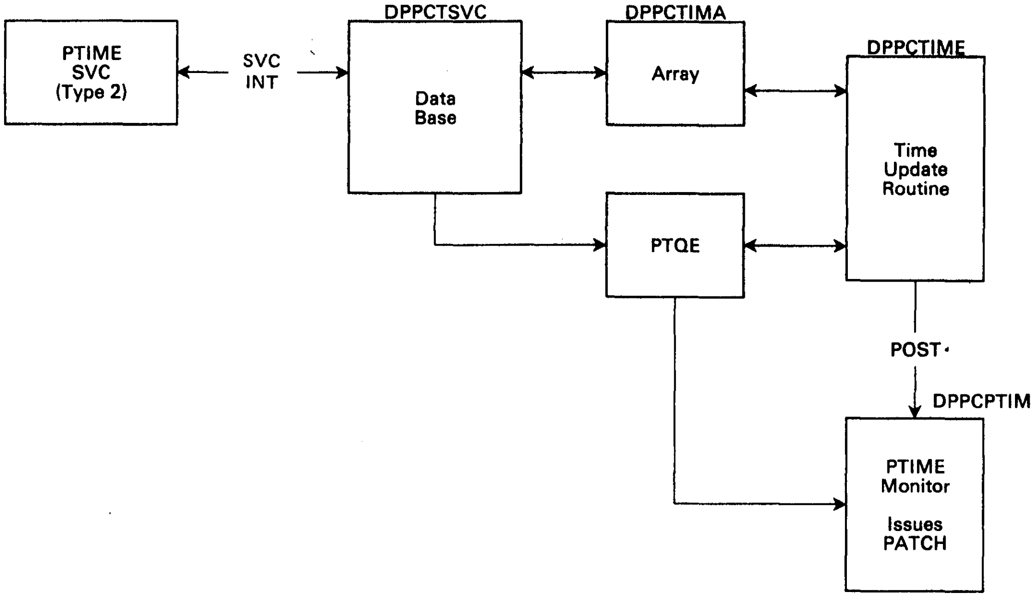


Figure 2-7. PTIME Logic and Control Flow

A Special Real Time Operating System data base array, DPPCTIMA, contains the Special Real Time Operating System time and date in several formats as shown below:

```
+TIMED      DSECT
+***
+*          TIME ARRAY DSECT
+***
+TIMEHS        DC   F'0'        TOD IN 10 MIL UNITS IN HEXADECIMAL
+TIMETOD       DC   F'0'        TOD IN DECIMAL 10 MIL UNITS-HHMMSSTH
+TIMEJDAY      DC   F'0'        JULIAN DATE-00YYDDDC
+TIMEMDAY      DC   F'0'        DAY OF MONTH DATE-0MMDDYYC
+TIMEEBC       DC   CL10'       EBCDIC DATE-bDD/MMM/YY
```

The Special Real Time Operating System time and date can be synchronized with an external time source or can be adjusted by manual inputs through customer-written interface programs. The Special Real Time Operating System time is updated at a periodic rate specified at the Special Real Time Operating System system build. A PTIME macro call will return the current Special Real Time Operating System time and the address of the Special Real Time Operating System time data base array. The address of the array can also be obtained from a pointer in the SCVT at label SCVTTIME or from a GETARRAY macro call for array name DPPCTIMA.

The Special Real Time Operating System time management facilities provide the ability to specify PATCHes which will be issued by a time management task at the requested time intervals. The PATCH operands (e.g., time of the first PATCH, interval between PATCHes) are defined in the PTIME macro call. The PATCH may be issued only once at a specified time or repeated for a specified number of PATCHes. Also the PATCH may be issued repeatedly at a specified time interval for an indefinite period of time. The PTIME macro call can also be used to modify or delete a previously defined PTIME.

There are three functional areas of the Special Real Time Operating System time management.

- The PTIME macro and the resulting PTIME SVC, DPPCTSVC, provide the user interface to time management.
- The time update routine, DPPCTIME, operates as an OS/VS task and is responsible for maintaining the current Special Real Time Operating System time in the data base array.
- The DPPCPTIM monitor routine, which also operates as an OS/VS task, is responsible for issuing the PATCHes requested via the PTIME macro call.

The time management programs are described individually in the following section.

PTIME Macro and PTIME SVC

The PTIME macro provides the user with an interface to the Special Real Time Operating System time management services.

PTIME can be used to cause a task to be given control at a given time, cyclically at a given interval, or cyclically at a given interval from time x to time y.

There are four types of PTIME service requests:

- RET -- This causes the system to return the current Special Real Time Operating System time in register 0 and the address of the Special Real Time Operating System time array in register 1.

  Note: Since the time contained in the array is updated only at a periodic rate, the time returned as a result of a PTIME RET macro call will be more exact than the array value.

- ADD -- This causes the system to build a PTIME queue element (PTQE) which exists independently of the creating task. This control block contains all information required to issue a PATCH macro; that is, the PATCH parameters are built according to the "PATCH operands" specified on the PTIME macro and are contained in the PTQE. The PTQE also contains information necessary for issuing the PATCH at the specified time; and, if requested, repeatedly reissuing the PATCH at a given time interval until the specified number of PATCHes has been issued or until a specified stop time has been reached. A PTIME ID may be supplied by the or assigned by Special Real Time Operating System if omitted by the user. The ID will be returned to the user in register 1.

  Note: If the interval time is omitted or if the interval time is less than the SYSGEN time interval used for updating the Special Real Time Operating System time array, the SYSGENed time interval will be substituted for the interval time.

- MOD -- This causes an existing PTQE to be modified. Since the PTQE exists independently of the creating task, the PTQE is referred to by a combination of task name, entry point name, and/or ID value of the parameter referred to by the operands TASK=, TASKLOC=, EP=, EPLOC=, and/or ID=. Either task name or entry point name must be specified, but the remaining two are optional. An additional level of identification, the PTIME ID, can be used to uniquely identify a PTQE eventhough several PTQE's may exist with the same PATCH parameters. However, if only a task name or an entry point name is specified on a PTIME MOD macro call, all PTQEs with that name are modified regardless of the original entry point name or task name, respectively.

- DEL -- This causes an existing PTQE to be deleted. Since the PTQE exists independent of the creating task, the PTQE is referred to by a combination of task name, entry point name, and/or ID value of the parameters referred to by the operands TASK=, TASKLOC=, EP=, EPLOC=, and/or ID=. Either task name or entry point name must be specified, but the remaining two are optional. An additional level of identification, the PTIME ID, can be used to uniquely identify a PTQE eventhough several PTQE's may exist with the same PATCH parameters. However, if only a task name or entry point name is specified on a PTIME DEL macro call, all PTQEs with that name are deleted regardless of the original entry point name or task name, respectively.

  For example, assume that a given user program were to be executed from a Special Real Time Operating System job step and assume that the given program contained the following macro calls:

```
ONE          PTIME        RET
TWO          PTIME        ADD,TASK=A,EP=X,ID=4,...
THREE        PTIME        ADD,TASK=A,EP=X,ID=5,...
FOUR         PTIME        ADD,TASK=B,EP=X,ID=5,...
FIVE         PTIME        MOD,TASK=A,EP=X,ID=4,...
SIX          PTIME        DEL,EP=X,...
                .
                .
                .
```

Macro call "ONE" causes the current time and the address of the Special Real Time Operating System time array to be returned to the user.

Macro call "TWO" causes a PTQE to be built so that PATCHes could be issued for task A, entry point X with an ID of 4.

Macro call "THREE" causes a PTQE to be built so that PATCHes could be issued for task A, entry point Y, with an ID of 5.

Macro call "FOUR" causes a PTQE built so that PATCHes could be issued for task B, entry point X, with an ID of 5.

Macro call "FIVE" causes the PTQE built by macro call "TWO" to be modified.

Macro call "SIX" causes the PTQEs built by macro call "TWO" and "FOUR" to be deleted.

Note: If the PTQE is specified by a combination task name, entry point name, and/or ID value cannot be located on a PTIME MOD or DEL, no action is taken by the system, and the user is notified of this condition by a return code of 8. That is, it had not been previously defined by a PTIME ADD, it had been deleted through a PTIME DEL, it had reached the specified STOP time, or it had issued the specified number of PATCHes.

The PTIME macro allows the user to specify a time to begin issuing PATCHes (START=), a time to cease issuing PATCHes (STOP=), or a total number of PATCHes to be issued (count=), and a time interval between PATCHes (INTERVAL=). All time values are specified in the same format. The time is specified explicitly by hours, minutes, seconds, or any combination of the three. The time value must not exceed 24 hours.

For example, if a relative start time of three hours is required, the PTIME macro could be coded in any of the following three forms:

```
 PTIME START=(3H),...
 PTIME START=(180M),...
 PTIME START=(1H,60M,3600S),...
```

If a relative stop time of 1 hour, 3 minutes, 1 and 1/2 seconds is required, the PTIME macro could be coded as:

```
 PTIME STOP=(1H,3M,1.5S),...
```

If four PATCHes are to be issued regardless of the start time, the PTIME macro could be coded as:

```
 PTIME COUNT=4,...
```

In addition to explicitly coding the time fields within the PTIME macro, the required time values may be loaded in a register or contained in a fullword at the address specified. However, the time values must be specified in binary hundredths of seconds to use either the register or address form of the PTIME macro.

For example, the following sequence of code

```
       .
       .
       .
  LA   3,5
  PTIME START=(A=ASTART),COUNT=(3),...
       .
       .
       .
  ASTART DC F'500'
       .
       .
       .
```

would cause five PATCHes to be issued with a relative start time of five seconds.

To allow greater flexibility in controlling the time of the PATCHes, three suboperands are permitted with the START= and STOP= keyword parameters of the PTIME macro.

- REL -- This suboperand is used to indicate that the time value is relative to the current Special Real Time Operating System time. That is, the time value in the keyword parameter is added to the current Special Real Time Operating System time to determine the correct start or stop time. This is the default suboperand.

- TOD -- This suboperand is used to indicate that the time value is time of day value. That is, the first PATCH will occur when the Special Real Time Operating System time is equal to the time of day specified in the remainder of the operand. If this time value is less than the current Special Real Time Operating System time, then the first PATCH will not be executed until the next day.

- ADJ -- This suboperand is used to indicate that the time value is an adjusted time of day value. That is, if the specified time value is less than the current Special Real Time Operating System time, then the time of the first PATCH is calculated by repeatedly adding the time value of the INTRVAL= operand to the specified time value until the sum is greater than the current Special Real Time Operating System time. This prevents the possibility of unintentionally specifying a TOD less than the current Special Real Time Operating System time and the first PATCH not occurring for almost 24 hours. If the specified time value is greater than the current Special Real Time Operating System time, the n processing would proceed as if the TOD suboperand had been coded.

Assume that the current Special Real Time Operating System time is 11:05, and the following PTIME macro call was executed:

```
ONE PTIME START=(TOD,10H),STOP=(ADJ,10H,30M),INTRVAL=(1H),...
```

PTIME macro call "ONE" would cause a PTQE to be built with a start time of 10:00. Since this is less than the current time, a 24-hour value is added to the start time so that the actual start time is 10:00 of the following day. The specified stop time (10:30) would be adjusted to 11:30 (i.e., 10:30 plus the interval of 1 hour). Since the stop time would be less than the start time, a 24-hour value is added to the stop time so that the actual stop time would be 11:30 the following day.

The remaining PTIME operands are identical to the PATCH operands, and their functions are described in the PATCH macro documentation. Two restrictions should be noted.

1. QPOS=DPATCH cannot be specified. LAST will be substituted.

2. FREE= Can be specidied, but the FREEMAIN will not be executed until the PTIME queue element (PTQE) generated by this PTIME is deleted. If the PTQE is not repeating, this will be like a normal PATCH.

Note: In response to a PTIME DEL request or a return code greater than 8 on the resulting PATCH macro call, the FREEMAIN will be executed when the PTQE is deleted, regardless of any outstanding work requests. This may result in abnormal termination of a program trying to reference the area that has been freed.

Time Update Routine

The time update routine executes under a high priority task and is in a continuous loop repeating at a rate specified during the Special Real Time Operating System system generation. Each execution causes the time value in the data base to be updated. The value retrieved from the System/370 TOD clock is adjusted by a conversion factor so that the Special Real Time Operating System time can be maintained independently of the OS time routine. The time update routine detects any inconsistency between the TOD clock and the Special Real Time Operating System time. If an inconsistency is discovered, a new conversion factor is calculated to correct the Special Real Time Operating System time.

After the current Special Real Time Operating System time has been calculated, the time update routine determines whether a PTQE requires servicing, and if so, the PTIME monitor routine is notified.

PTIME Use of the Clock Comparator

The PTIME time update routine normally controls its execution rate by issuing STIMER to delay for the specified amount of time. Optionally the PTIME time update routine can be directed to use the optional clock comparator feature of the System/370, if OS/VS1 is generated to not use this feature. This feature is selected by coding CLOCKCP=YES in the VS macro of the Special Real Time Operating System SYSGEN. PTIME usage of the clock comparator, if selected, is available to the first single partition real-time job step that enters the system or to the first "MASTER" partition to enter. Use of the clock comparator saves STIMER overhead. If other Special Real Time Operating System real-time jobs are also run at the same time, they will use the STIMER interface.

PTIME Monitor Routine

The PTIME monitor routine is responsible for issuing PATCHes requested via a PTIME macro call. All active PTQEs with the time of the next PATCH less than the current time plus the SYSGENed update interval are serviced by issuing a PATCH. If the PTQE is repeating and the count of PATCHes has not been exceeded, the next PATCH time is calculated; otherwise, the PTIME request is terminated, and the PTQE is deleted.

Time Drift Correction

Many real-time systems require highly accurate maintenance of time of day. The System/370 TOD clock is susceptible to a certain amount of drift. As a result, a user may wish to correct this drift by using a highly accurate external time source to correct for TOD clock drift. The time drift correction feature of the Special Real Time Operating System allows for correction of long term drift in the System/370 TOD clock. Time drift correction is optionally selected during the Special Real Time Operating System SYSGEN if support is required for an external time source. To include time drift correction in the Special Real Time Operating System, the TIMEEXT keyword on the VS macro of the Special Real Time Operating System SYSGEN must be coded to specify the external signal line (2-7) on which the time interrupt will occur. The external time source may then interrupt the Special Real Time Operating System at the given frequency and allow for correction.

Time drift correction operates as a Special Real Time Operating System subtask with a module name DPPDRIFT. The feature operates by accepting external interrupts from the external source on a periodic basis from one per second to one per ten minutes. A period of one interrupt per minute is recommended. To create a time interval of other than one minute, the required value must be specified in the TIMERAT keyword of the VS macro during the Special Real Time Operating System SYSGEN.

The external interrupts are assumed to be accurate. If the external time standard is not accurate, the TOD clock will appear to have excessive drift. An allowance is made for discrepancies caused by delay in the handling of the interrupt. This type of delay can occur if the interrupt arrives at a point in time when the CPU is disabled for external interrupts.

Drift corrections are made by passing adjustment factors to the Special Real Time Operating System time routines which update the Special Real Time Operating System time conversion factor, not by altering the TOD clock. Time drift correction does not supply any initial times, it merely accounts for long term drift. The function of passing an adjustment factor and the functional relationship between time drift correction and the Special Real Time Operating System time management is illustrated in Figure 2-8 below.

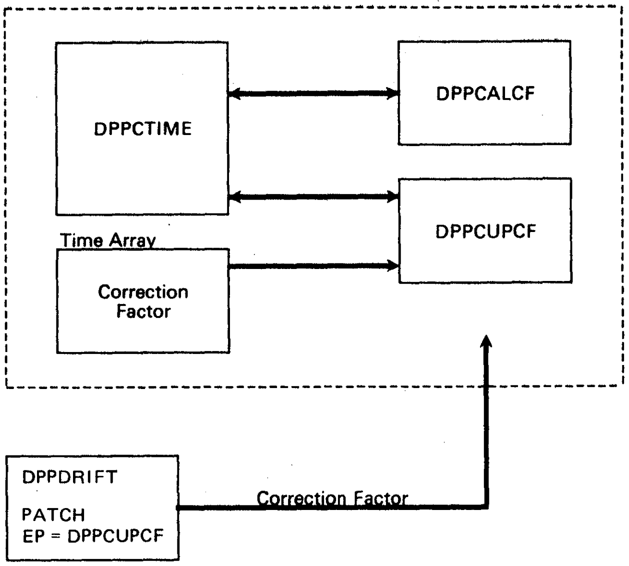


Figure 2-8. Time Drift-Special Real Time Operating System Time Relationship

The maximum correction made at one instant is 50 milliseconds; the minimum is 10 milliseconds. Errors of greater than 50 milliseconds are spread over succeeding corrections until the time error has been corrected. Small differences in which the TOD clock appears fast are not corrected immediately, as the difference may be due to processing or interrupt lockout time. These errors are averaged over many interrupts before the correction is made. Errors in which the TOD clock appears to be behind are always adjusted immediately. Interrupts indicating errors in excess of one second are ignored. Resetting of the Special Real Time Operating System time (PTIME) by an application program has no effect on drift accounting. The resetting of the TOD clock by a user program, however, will cause unpredictable results. A malfunction in the TOD clock that causes condition code settings of 2 or 3 on an OS STCK instruction will cause the termination of time drift correction.

Time drift correction supplies a user interface to allow a user's program to set current time. The first external signal time interrupt following the completion of initialization causes a LINK to DPPDRIFE. On entry to DPPDRIFE, general register 1 points to a doubleword containing the value of the TOD clock at the time of the external time interrupt. The module DPPDRIFE supplied by the Special Real Time Operating System is a dummy, and the user may replace with a module to set initial time. Using standard OS linkage conventions, DPPDRIFE can issue a PATCH to the Special Real Time Operating System time management to adjust the conversion factor (see Figure 2-8). The format of the data to be passed is described in the Special Real Time Operating System Program Logic Specification.

Warning: By whatever means the user version of DPPDRIFE determines the desired system time, the user must be aware that the time of its determination is some time later than the time stored at interrupt time. The amount of delay can be determined by reading the TOD clock and subtracting from it the value passed as a parameter (pointed to by register 1).

Drift correction is available to the first single partition Special Real Time Operating System real-time job that enters the system or the first "MASTER" partition to enter. If more than one Special Real Time Operating System is run on the same OS/VS1 system, time correction will be suppressed for the other Special Real Time Operating System systems. Thus for testing purposes, an application should be coded such that its DPPDRIFE routine does not have to be executed for the application to function. Time drift correction is never available to "SLAVE" partitions, as SLAVE partitions use their MASTER partitions time management tables.

REAL TIME MESSAGE HANDLER

The Special Real Time Operating System provides facilities for defining a series of messages by means of an offline utility program. These messages can later be modified and issued in real-time. All messages can be predefined and kept in a partitioned data set on a direct access storage device. This allows for easy modification of messages without making changes to functioning programs. It also allows for easy translation of all messages to other languages and avoids duplication of messages. The data set is created and updated by the offline utility program (DPPXUTIL). It is used online only as input to the message writer. Although this data set is built by the offline utility, it is a normal partitioned data set and none of the data base data set restrictions apply to it. There are two components to the real-time message handler: offline processing and online message processing. This is illustrated in Figure 2-9.

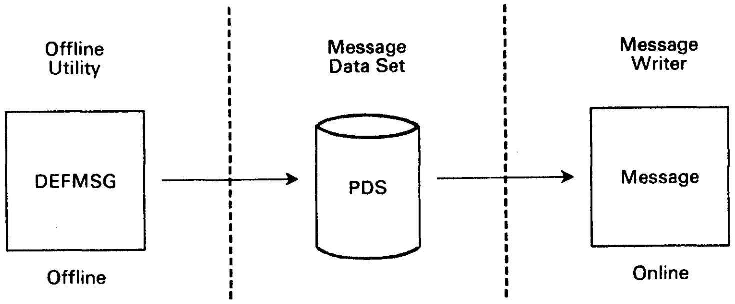


Figure 2-9. Real Time Message Handler Components

Offline Processing

The DEFMSG macro is used to define messages to the offline utility program that processes the macro and places the resultant skeleton message in the message data set. For further information on the offline utility, refer to the section entitled "Offline Utility Program". The DEFMSG macro defines a unique message number, the routing code, action code, a date indicator, and the message text.

The message number identifies a specific message and is the means by which online programs refer to that message. The message number has a range from 001-999. The Special Real Time Operating System messages fall within the range of 001-099 and 800-899. The user should not assign message numbers in these ranges. Related PRPQs should restrict their messages to a defined range. This is by convention only, and no restrictions are placed on the user's message numbers.

The routing code (ROUTE=) is used to specify the output device to which the message is to be written. At the Special Real Time Operating System SYSGEN time, routing codes are established to identify the output device

The routing code has a range of 1-255. It can also identify a user program as a device, in which case the message is passed to the program as a PATCH parameter. A routing code must be specified with the DEFMSG macro. A routing code of 255 results in a no-operation (255 goes to no output device).

The action code (ACT=) identifies the type of action that the message requires. ACT=I identifies the message as being informational only. ACT=A means that some action is required. ACT=D requires a decision to be made. These codes cause no action within the message output process but are intended for user information.

The date indicator (DATE=) is the date that the message was issued from an online program. The date can be included (DATE=YES) or excluded (DATE=NO). DATE=NO is the default.

The message text (TEXT=) contains the text of the message to be written or passed to a PATCHed program. Within the text, there can be variable data. The variable data will be inserted when the message is issued online. Variables are specified to appear in the message by coding, in the message definition, information in the following format:

```
   #cfs#
```

where: # (pound sign) is a delimiter character and must appear before and after the other specifications. No blanks are allowed between them.

c defines the number of characters to be occupied by this variable in the output message.

f defines the type of data conversion to be performed on the data being output.

s specifies the position of this variable in the variable list that is passed by the calling program when the message is selected for output.

The following are examples of the use of the DEFMSG macro.

```
EXAMPLE 1: DEFMSG 307,ROUTE=10,ACT=I,DATE=YES,TEXT='THIS MESSAGE HAS
           NO VARIABLES'
```

This defines message 307 as being informational; a routing code of 10, and the time and date which are to be inserted when the message is to be written.

```
EXAMPLE 2: DEFMSG 2,ROUTE=250,ACT=D,DATE=NO,TEXT='PROGRAM #8C1#
           HAS TERMINATED.  SHOULD PROGRAM #8C2# CONTINUE?'
```

This defines message 2 which has a routing code of 250. The date will not be formatted in the message, and the text contains two character variables.

```
EXAMPLE 3: DEFMSG 50,ROUTE=1,ACT=D,TEXT='MSG #3C3# HAS FIVE VARIABLES:
           #2F1#, #1H2#, #6B4#, #5X5#'
```

Message 50 will require a decision, has a routing code of 1, will not print the date, and has five variables:

1. #2F1# is the first variable with a length of 2 characters and integer format, and the user will provide a fullword for conversion.

2. #1H2# is the second variable with a length of 1 character, integer format, and the user will provide a halfword for conversion.

3. #3C3# is the third variable with a length of three characters, and the user will provide 3 EBCDIC characters to be inserted.

4. #6B4# is the fourth variable with a length of 6 characters, binary format, and the user will provide one byte for conversion (the six low order bits of the byte will be converted).

5. #5X5# is the fifth variable with a length of 5 characters, hexadecimal format. The user will provide 3 bytes of data for conversion (the five low-order hexadecimal digits will be converted).

Online Processing

Messages are retrieved, formatted, and written during online processing through the MESSAGE macro. With the MESSAGE macro, options selected by the DEFMSG macro can be overridden, or omitted from the MESSAGE macro, and the DEFMSG options taken. The AREA= operand will indicate that the message is to be returned to the user specified area. The area should be defined at least to the maximum length of the message plus two bytes. The length of the message is put into the first two bytes of the virtual storage specified by AREA= and the formatted text in the remaining bytes.

The maximum message length that can be moved is 255 characters.

If the message contains variables, the user passes the data to be inserted in the message (VAR=). The data is inserted in the order presented into the variables fields defined by the DEFMSG macro (see examples below).

In online processing, a message can be output to several devices by two methods. The MESSAGE macro allows up to 8 routing codes to be specified and the MSGRC macro of the Special Real Time Operating System SYSGEN can be included, for a given routing code, several times, each time specifying a different device.

If a message is issued to a routing code that does not exist, no attempt is made to output the message, and a return code of 12 is returned to the user. When a message is issued to multiple devices, and one of the devices is out-of-service, an attempt will be made to issue the message to the backup (alternate) device defined during SYSGEN. The out-of-service route code does not affect the other route codes. The message will still be output to these devices.

The format of the message is an identifier, time, and date (if requested), and text. The identifier is:

```
DPPnnna

     where:     nnn is message number
                  a is the action code
```

The time and date are represented by:

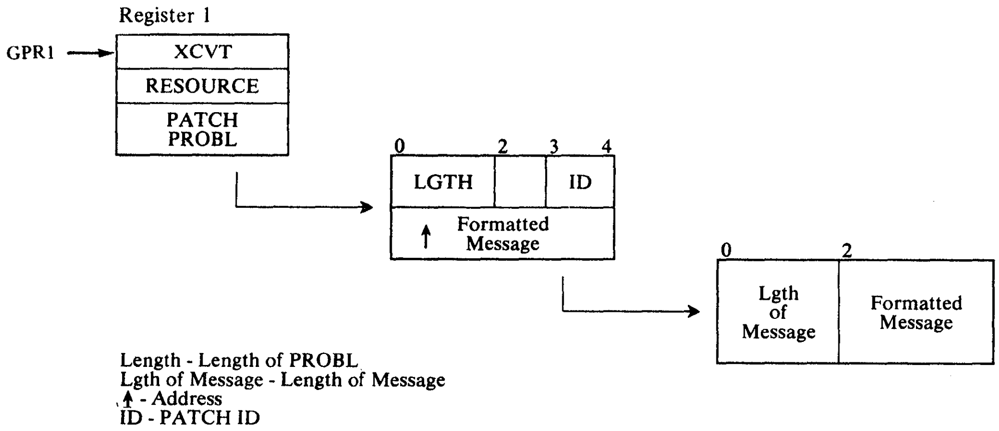
```
HH:MM:SS.t      DD/MMM/YY

     where:     HH is hour
                MM is minutes
                SS is seconds
```

  t is tenths of seconds
MMM is month
 DD is day
 YY is year

The message will be truncated to conform to the line length of the device selected by the routing code.

When a message is routed to a user program, the PATCH parameters and the message will be in the following format.



Note: All messages issued prior to the processing of a RESTART card and during initialization will be written to the system console. After the RESTART card is processed or if there was no RESTART card, the messages will be routed to their respective routing code devices.

The following examples of the MESSAGE macro show the resulting messages for the previously defined DEFMSG macro.

EXAMPLE 1: MESSAGE 307,ROUTE=(1,2,3),ACT=A,AREA=MSG. In this example, message 307 will be routed to the devices identified by routing codes 1, 2, and 3. The routing code on the MESSAGE macro overrides the ROUTE= from the DEFMSG macro. The formatted message will be returned to the user area labeled MSG. The resultant message will appear as follows:

```
  DPP307A 14:37:21:92 07/JAN/73 THIS MESSAGE HAS NO VARIABLES
```

EXAMPLE 2: MESSAGE 2,VAR=((7),(8)). In this example, message 2 will be routed to the device or program for routing code 250. The date will not be printed. The message will require a decision. The registers (7 and 8) point to areas in virtual storage from which eight characters will be moved into the message variables before the message is written. Assuming that register 7 points to the eight characters TIMECALL, and register 8 points to the eight characters CORRFACT, the resultant message would appear as follows:

```
 DPP002D 12:22:20:21 PROGRAM TIMECALL HAS TERMINATED.
   SHOULD PROGRAM CORRFACT CONTINUE?
```

EXAMPLE 3: MESSAGE 50,ROUTE=(21,1),ACT=I,VAR=(A,B,C,D,E). In this example, message 50 will be routed to the devices or programs specified by 21 and 1. The message consists of information, overriding the DEFMSG action A. The date will print as YES on the default. Assuming the following pointers:

```
A = fullword integer = DC F'320'
B = halfword integer = DC H'9'
C = character DC C'004'
D = binary integer DC B'011011'
E = hexadecimal DC X'CA420'
```

The resultant message would be:

```
DPP050I 14:39:20:07 07/FEB/71 MSG 004 HAS FIVE VARIABLES:
  320,09,011011,CA420.
```

Message Routing Code Status Change Facility

The Message Routing Code Status Change Facility provides a service which allows the user to place a routing code in or out of service. The facility will also provide upon request the status of one or all of the routing codes in the Special Real Time Operating System message handler.

The facility is activated by an Input Message Processing (IMP) command.

The format of the command is:

```
MSGRC,{rc,} {IN     }
      { 0 } {OUT    }[,altrc]
            {STATUS }
            {STATALL}
```

MSGRC Informs the IMP routine that this reply is for the Message Routing Code Status Change Facility.

rc Routing code.

0 This parameter is 0 if STATALL is specified.

IN Place rc in service.

OUT Place rc out of service.

STATUS Display the status, via a system message, of the specified routing code (rc).

STATALL Display the status, via a system message, of all the routing codes in the system.

altrc The routing code to which messages are directed should the primary routing code be out of service or the output operation fail. This parameter is recognized only if IN or OUT is specified.

REPORT DATA OUTPUT FACILITY

A facility is provided to transfer report data which is ultimately destined to be printed, from one or more working data sets to a QSAM supported output data set.

The Report Data Output Facility will write the data as it is generated to working data sets (QSAM data set on any QSAM device). Subsequently, the data may be transfered to a print device. The data could be collected from several working data sets by the report data output facility and written to another data set to be printed by a job step in another partition or another computer which shares direct access (DA) devices with the online computer.

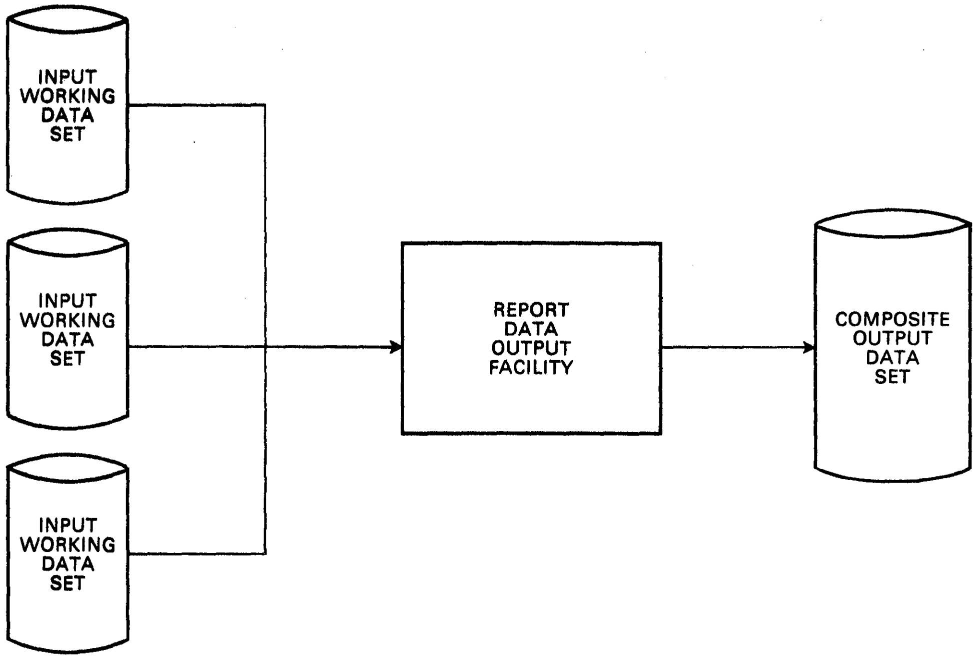


Figure 2-10. Report Data Output Facility Overview

All input and output data sets used by the Report Data Output Facility must be BSAM data sets. The maximum record length must not be greater than 255. For a unit record device the BLKSIZE and LRECL must not exceed the maximum for that device. The Report Data Output Facility is invoked through IMP commands.

```
                  [{ ADD }]
REPORT,[SLAVE,]   [{ NEW }] ,OUTPUT DDNAME, INPUT DDNAME,
[INPUT DDNAME,...,INPUT DDNAME]
```

REPORT
Informs the input message processing routine that this reply is for the Report Data Output Facility.

SLAVE
Indicates the PATCHed routine is to run in the SLAVE partition.

NEW
Report Data Output Facility starts writing data at the beginning of the output data set.

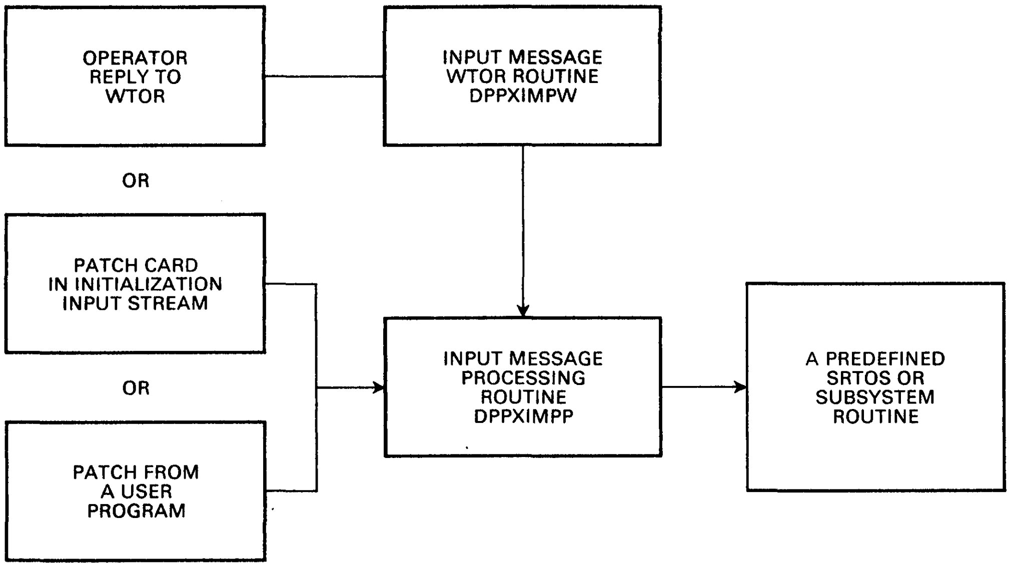
ADD
Report Data Output Facility adds all data at the end of the output data set.

OUTPUT DDNAME
A DD name which points to a QSAM data set to be used as the output data set. The BLKSIZE of the data set must be equal to or greater than the maximum BLKSIZE of the input data sets.

INPUT DDNAME
A DD name which points to a QSAM data set to be used as an input data set. A maximum of 10 input DD names may be specified.

INPUT MESSAGE PROCESSING

The Special Real Time Operating System provides a facility to allow for operator--Special Real Time Operating System communication or for the operator to communicate with a subsystem. This facility is the Input Message Processor (IMP). The Special Real Time Operating System, during initialization, issues a WTOR and leaves the reply outstanding. At a later time, the operator may reply with a predefined IMP command. This IMP command is defined at SYSGEN by the IMP macro and also defines the action the Special Real Time Operating System is to take upon receiving the IMP code. The following example shows the sequence of events and alternate methods of invoking Input Message Processor.


OPERATOR REPLY TO WTOR
INPUT MESSAGE WTOR ROUTINE DPPXIMPW
OR
PATCH CARD IN INITIALIZATION INPUT STREAM
OR
PATCH FROM A USER PROGRAM
INPUT MESSAGE PROCESSING ROUTINE DPPXIMPP
A PREDEFINED SRTOS OR SUBSYSTEM ROUTINE

Input Message Processing will accept IMP commands in the following format: "code,param1,param2,...,paramn" where code is the command word defined during SYSGEN by the IMP macro. Param corresponds to the parameters defined by the IMP macro. Any parameters may be omitted by entering double commas (null parameters). The command will be compared with entries in a table (an array in the data base). This table contains valid IMP commands, the names of the task and load module which process the command (the program to be PATCHed), PATCH ID, and parameter conversion codes. If the IMP command is valid, Input Message Processing will patch the appropriate task with the specified input parameters.

New commands can be added to the table through SYSGEN. Input Message Processing will accept commands from several different sources (as a reply to a WTOR, through a PATCH macro and initialization PATCH Input Cards). The different ways of entering IMP commands are described below. The keyword SLAVE in all cases is optional; and if omitted, should not have the comma included to represent its absence.

- Input Message Processing will issue the following WTOR: 'Input Message Processing Awaiting Reply'. In response to this WTOR, the operator can issue an IMP command. There will always be an outstanding WTOR in the system. In response to an IMP command, Input Message Processing will issue the following message (WTO).

  ```
  IMP COMMAND RECEIVED.
  ```

  The IMP commands are in the following format:

  ```
  r xx,command,SLAVE,param1,param2,...,paramn
  ```

where r xx, is the format required by OS/VS.

- IMP commands issued through initialization PATCH input cards must be in the following format:

```
P1 PATCH        EP=DPPXIMPP,ID=0
                PARAM=(C'command,SLAVE,param1,param2,...,paramn
```

EP=DPPXIMPP The entry point of the input message processing routine. No TASK= parameter is specified because the task must be dependent.

ID=0 The ID must be 0.

PARAM= (C'imp command') is the IMP command to be processed.

SLAVE The command is to be processed in the SLAVE partition.

- IMP commands issued through a PATCH macro must be in the following format:

```
                L         r,ADDR

    P2          PATCH     EP=DPPXIMPP,ID=1,PARAM=((r))

    ADDR        DC        AL1(LGTH),AL3(IMPCODE)

    IMPCODE     DC        C'CODE,SLAVE,param1,param2,...,paramn
```

where:

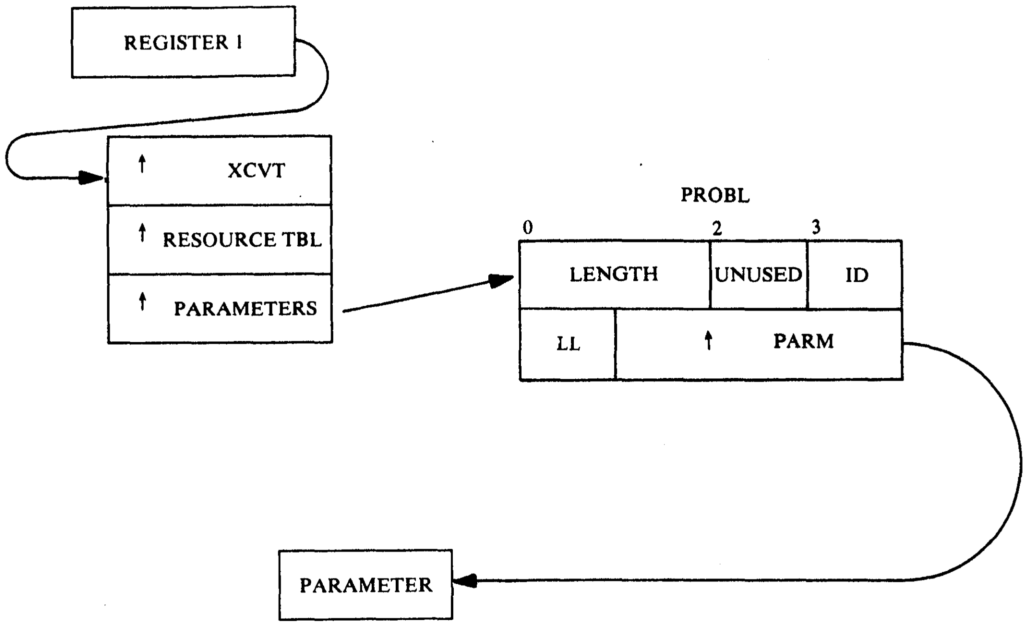
ID = 1 The ID must be 1 when entered through PATCH macro.

ADDR This is a 4-byte area. The first byte contains the length of the IMP command and the next three bytes contain the address of the IMP command.

r Register 2-12

IMPCODE The IMP command.

The following example shows the parameters as they would appear when the task which is patched as a result of the IMP command gains control. If no parameters are passed, there will be no parameter pointer. A null parameter results in a zero address being passed for the parameter address.



↑ = The address of a parameter.
LENGTH = Length of PROBL plus PARMS.
ID = ID specified during SYSGEN.
LL = Length of this parameter.
PARM = ADDRESS of parameter.

Note: The first LL and PARM parameters may contain zeros if only the last parameters of a multiparameter IMP code are specified, example:

'code,,,param3, param4'.

When only the first parameters of a multi-parameter IMP code are specified, the last parameters defined during SYSGEN by IMP macro will be ignored. A comma followed by a comma(,,) with no intervening character constitutes a null parameter.

EXAMPLE 1: This example shows an IMP command being defined:

```
 SYMBOL IMP    CODE=EXAMPLE1,TASK=DPPTEST,
               LM=DPPTEST,ID=0,
               PARAM=(C10,F4,X3)
```

In this example, an IMP command is defined with a command word of EXAMPLE1. DPPTEST will accept three parameters:

1. a character parameter of length 10.

2. a fullword parameter of length 4.

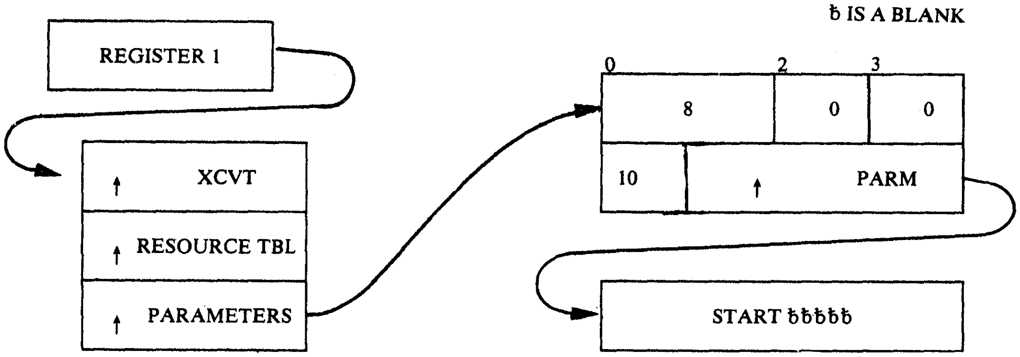
3. a hexadecimal parameter of length 3.

For more details on defining IMP commands see the section on SYSGEN macros (IMP macro).

EXAMPLE 2: In this example, the IMP command defined in EXAMPLE 1 will be entered through the system console as a reply to the WTOR "INPUT MESSAGE PROCESSING WAITING ON REPLY".

```
r xx,'EXAMPLE1,SLAVE,START'
```

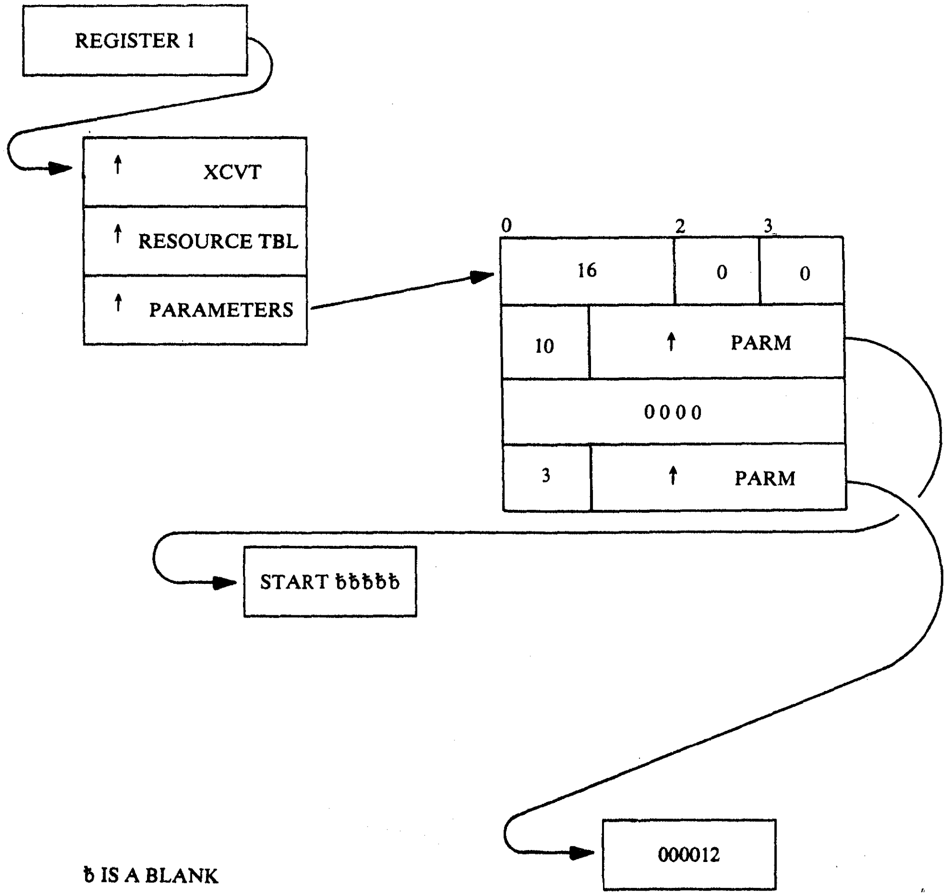
SLAVE parameter says DPPTEST is to execute in the SLAVE partition. When DPPTEST is entered, the parameters will be in the following format:



EXAMPLE 3: In this example, the IMP command defined in EXAMPLE 1 will be entered through the initialization input stream.

```
PATCH  EP=DPPXIMPP,ID=0,
  PARAM=(C'EXAMPLE1,START,,12')
```

When DPPTEST is entered, the parameters will be in the following format:

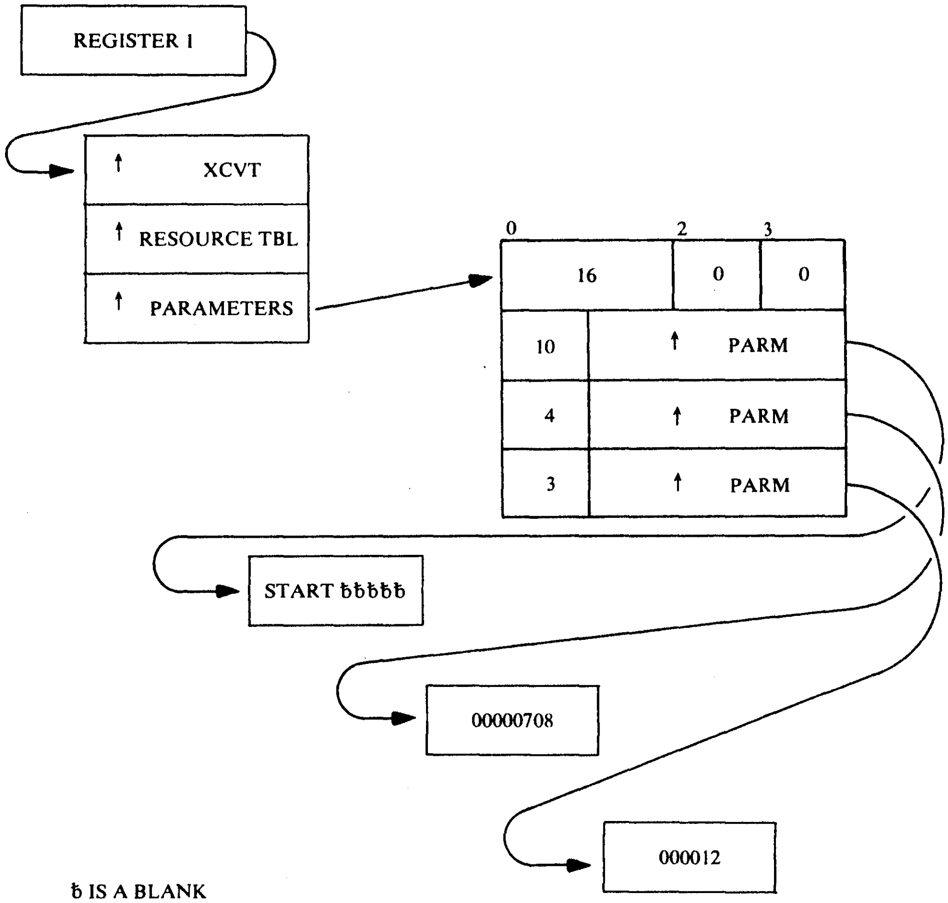


EXAMPLE 4: In this example, the IMP command defined in EXAMPLE 1 will be entered by a PATCH macro.

The IMP code follows:

```
L    r,ADDR
P1  PATCH   EP=DPPXIMPP,ID=1,PARAM=((r))
     .
     .
     .
ADDR    DC   AL1(21),AL3(IMPCODE)
IMPCODE DC   C'EXAMPLE1,START,708,12'
```

When DPPTEST is entered, the parameters will be in the following format:



ƀ IS A BLANK

DATA BASE MANAGEMENT

The Special Real Time Operating System data base is designed to fulfill the needs of data storage and access of a realtime operating system. The Special Real Time Operating System data base subroutines provide the user with an interface to the information contained in the data base. Through the use of these subroutines, data may be retrieved from or replaced in the data base. In addition, sections of the data base may be copied to a direct access device to provide an historical log.

The data base consists of data items which are logically grouped into arrays. These arrays may also contain one or more blocks of related information. Each block is identical in size and shape to every other block within that array. For example, assume that the temperature and volume are to be monitored for three separate storage tanks. The two items (temperature and volume) can be grouped into one block. Three blocks (one for each storage tank) can be grouped into one array. This array can then be logged on a cyclic time interval to provide a history of the contents of the storage tanks as shown below.

Block 1 Storage Tank 1	Item A - Temperature Item B - Volume
Block 2 Storage Tank 2	Item A - Temperature Item B - Volume
Block 3 Storage Tank 3	Item A - Temperature Item B - Volume

The Special Real Time Operating System arrays can either reside in VS or on a DA device. Duplicate data set support will be provided for all data base data sets (i.e., data sets containing DA resident arrays). However, it is the user's responsibility to ensure that the data base data sets do indeed meet the requirements for duplicate data set support, to create the required backup data set(s), and to identify these data sets through the normal duplicate data set input stream (refer to the section entitled "Duplicate Data Set Support" for a detailed description of duplicate data set). VS resident arrays may either be blocked or nonblocked arrays and are eligible to be logged. All DA resident arrays must be blocked and cannot be logged. An array that contains a copy (or copies) of a loggable VS resident array is called a log array. All log arrays must be DA resident. All arrays must be defined by the offline data base utility which is discussed in detail in the section entitled "Offline Utility Programs."

The data base utility builds two data sets: (1) a data base initialization data set containing all the information necessary for the online data base initialization routine to construct the required control blocks, and (2) a composite items data set containing all the information necessary for the online data base subroutine to locate a particular item or items.

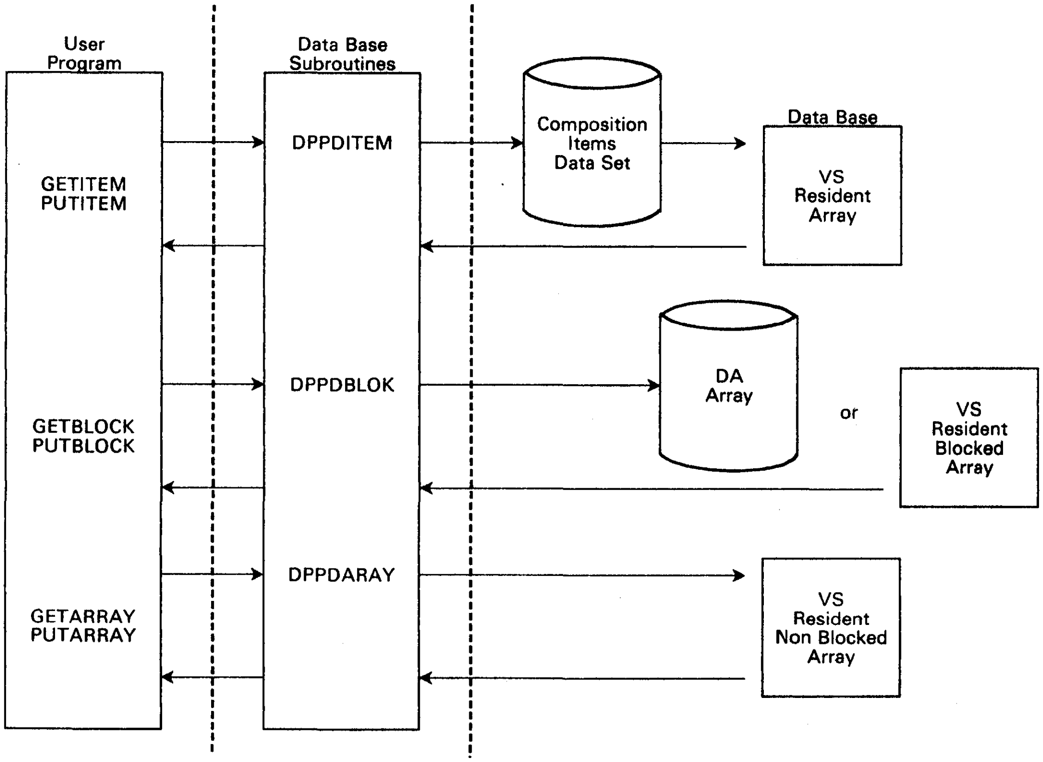
During a normal start, i.e., when the job is initially started through standard OS/VS1 Job Control statements with the EXEC card specifying PGM=DPPINIT, the data base initialization program will read in the initial data for all VS resident arrays that specified "INIT=YES" on the ARRAY macro in the offline utility phase. Those VS arrays for which "INIT=YES" was not specified have VS storage space allocated, but no data is moved into the space.

During a refresh start, i.e., when the job is reinitialized from a restart data set, or during a normal start when the SYSINIT input stream does not contain a "DBREF NO" control statement, the data base initialization program will refresh all VS resident arrays that specified "REINIT=YES" and that requested logging in the offline utility phase with the last logged copy of that array. The log arrays are initialized to resume logging with the last logged copy of each loggable VS resident array.

Note that VS resident arrays are arranged in virtual storage by the USE code specified during offline utility processing. Arrays with similar USE codes are grouped together in virtual storage. This is intended to optimize the use of real storage by improving the probability that the high usage arrays will remain in real storage. Grouping high usage arrays will cause them to be distributed in a smaller number of pages to reduce the number of page faults.

Data Base Access Routine

Access to the data base is achieved through a set of six macros: GETITEM, PUTITEM, GETBLOCK, PUTBLOCK, GETARRAY, and PUTARRAY as shown in the following example.



GETITEM
The GETITEM macro can be used to retrieve certain information from one or more items in the data base. This information is stored in the address indicated by the DATA= keyword parameter. The user may request that the address within the data base of the item(s) and length of the item(s) be retrieved (TYPE=ADDR) or that the data contained in each item be returned (TYPE=DATA). TYPE=DATA and TYPE=ADDR are valid for direct access resident arrays. For blocked arrays, the user must specify the number assigned to the data block which contains the item (BLKN=number). The item or items for which information is to be retrieved is indicated with the NAME=, NAMELST=, or ADDRLST= keyword parameter. The NAME= keyword parameter is an 8-character name of a single item for which information is to be retrieved. The NAMELST= keyword parameter specifies the address of a list of 8-character item

names for which information is to be retrieved. The ADDRLST= keyword parameter specifies the address of a list of data base item addresses which were returned from a previous execution of this macro with NAME= or NAMELST= specified and TYPE=ADDR. The PROTECT= keyword parameter allows the user the option (PROTECT= YES) of preventing other programs from modifying the data base during the execution of this GETITEM. If PROTECT=RISK is specified, the information will be moved without regard to other programs which may be storing into the data base.

The following examples indicate how the GETITEM macro may be used to retrieve information.

```
        GETITEM     NAMELST=A,TYPE=ADDR,DATA=B

        GETITEM     ADDRLST=B,DATA=C,TYPE=DATA,PROTECT=YES...

 A      DC          CL8'ITEM1'
 A1     DC          CL8'ITEM2'
        DC          X'FF'
 B      DC          A(0)
 B1     DC          A(0)
        DC          4X'FF'
 C      DC          CL16' '
 C1     DC          C132' '
```

The first GETITEM will move the length and address of items ITEM1 and ITEM2 into the data fields B and B1, respectively. The second GETITEM will move the data associated with the items whose addresses are contained in the address list fields, B and B1, into the data fields, C and C1, respectively. Therefore, data associated with ITEM1 will have been moved into C, and data associated with ITEM2 will have been moved into C1.

PUTITEM

The PUTITEM can be used to store data into one or more items of the data base. This data is moved from the address indicated by the "DATA=" keyword parameter. For blocked arrays, the user must specify the number assigned to the data block which contains the item (BLOCKNO=number). The item or items for which data is to be stored is indicated with the NAME=, NAMELST=, or ADDRLST= keyword parameter. The NAME= keyword parameter is an 8-character name of a single item for which data is to be stored. The NAMELST= keyword parameter specifies the address of a list of 8-character item names for which data is to be stored. The ADDRLST= keyword parameter specifies the address of a list of data base item addresses as returned from a previous execution of a GETITEM macro with a NAME= or NAMELST= specified and a TYPE=ADDR.

GETBLOCK

The GETBLOCK macro can be used to retrieve one or more data blocks from one or more blocked arrays. The arrays may be either VS or DA resident arrays. The NAME= and NAMELST= keyword parameters are used to indicate the 8-character name or names of the arrays from which one or more blocks of data are to be retrieved. The NUMBER and NUMBLST= keyword parameters are used to indicate the two-byte number or numbers assigned to a numbered array or arrays from which one or more blocks of data are to be retrieved. The DATALST= keyword parameter specifies the address of a list of block numbers and associated memory addresses where the data blocks are to be written. Each entry in the list will contain a byte flag field, a 3-byte area address, and a 2-byte block number. A flag byte of X'40' indicates the last entry to be processed for a particular entry in the name list or number list.

The PROTECT= keyword parameter allows the user the option (PROTECT=YES) of preventing other programs from modifying the data base during the execution of this GETBLOCK. For DA resident arrays, a PROTECT=YES request will reserve the data set containing the specified array. For VS resident arrays, the VS resident data base is reserved. If PROTECT=RISK is specified, the information will be moved without regard to other programs which may be storing into the data base.

For an example of the use of the GETBLOCK macro, assume that array FIRST is a VS resident blocked array and array SECOND is a DA resident array. For this example, each array is assumed to be composed of three 40-byte blocks. If the following GETBLOCK macro were to be executed, blocks 1 and 3 of the array FIRST would be moved into the DATA1 and DATA2, respectively.

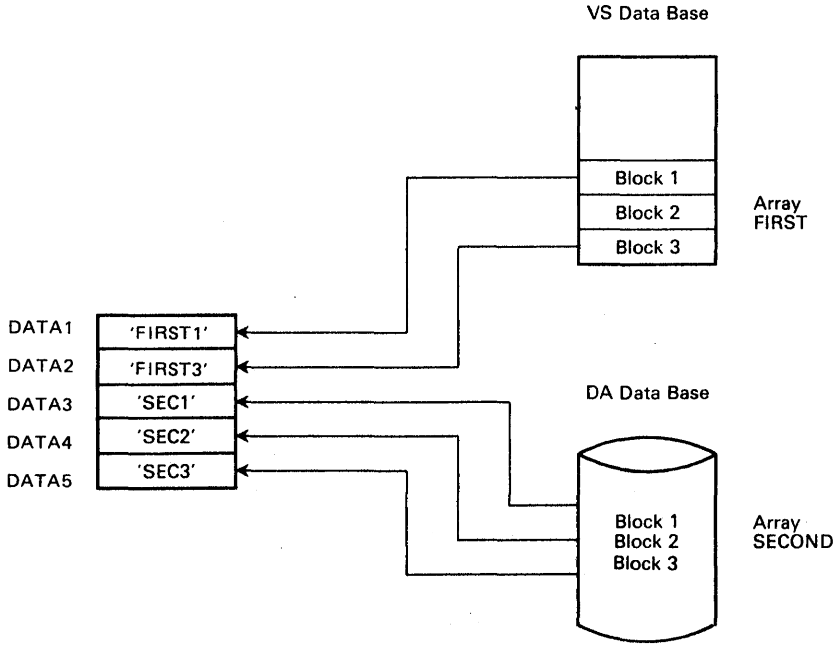
The entire array SECOND (blocks 1, 2, and 3) would be read into DATA3, DATA4, and DATA5, respectively.

```
          GETBLOCK     NAMELST=A,DATALST=B,...

 B        EQU          *
          DC           X'0',AL3(DATA1),H'1'
          DC           X'40',AL3(DATA2),H'3'
          DC           X'00',AL3(DATA3),H'1'
          DC           X'0',AL3(DATA4),H'2'
          DC           X'40',AL3(DATA5),H'3'

 A        EQU          *
          DC           CL8'FIRST'
          DC           CL8'SECOND'
          DC           X'FF'

 DATA1    DC           10F'0'
 DATA2    DC           10F'0'
 DATA3    DC           10F'0'
 DATA4    DC           10F'0'
 DATA5    DC           10F'0'
```



PUTBLOCK

The PUTBLOCK macro can be used to move data from one or more user specified virtual storage locations into one or more blocks of one or more blocked arrays. The arrays may be either VS or DA resident arrays. The NAME= and NAMELST= keyword parameters are used to indicate the 8-character name or names of the arrays into which one or more blocks of data is to be written.

The NUMBER= and NUMBLST= keyword parameters are used to indicate the two-byte numbers assigned to a numbered array(s) into which one or more blocks of data are to be written. The DATALST= keyword parameter specifies the address of a list of block numbers and associated storage addresses from which data blocks are to be written.

Other routines executing data base requests with a PROTECT=YES option will be prevented from accessing the VS resident data base (or DA data set) during the execution of a PUTBLOCK request.

GETARRAY

The GETARRAY macro can be used to retrieve data which is stored in VS resident array(s), to retrieve the address of and certain information about VS or DA resident array(s), or to determine specific information about all items defined as part of VS or DA resident array(s). Which type of data is to be retrieved is specified by the TYPE parameter. The array for which data is to be retrieved is identified through the NAME, NAMELST, NUMBER, or NUMBLST keyword operands. The NAME= parameter specifies the 8-character name of the array as defined through the offline utility data base definition. The NUMBER= parameter specifies the number (1-255) of the array. Associated with the NAME= or NUMBER= parameter, the DATA= parameter specifies the address to which the data is to be moved.

The NAMELST= parameter specifies the address of a list of 8-character names of one or more arrays for which data is to be retrieved. The NUMBLIST= parameter specifies the address of a list of one or more halfwords which contain the numbers which identify the arrays for which data is to be retrieved. The area(s) into which data is to be moved when NAMELST or NUMBLST is specified are identified by the DATALST or FINDLST parameters.

The data to be returned is specified by the TYPE= parameter. If TYPE=DATA is specified, the content of the entire array(s) is moved into the area specified by the DATA= or DATALST= parameters. If TYPE=SPEC is specified, the specification information (16 bytes) is returned for each item contained in the specified array(s). This information contains, for each item, item name, length of the item, defined data type, displacement into the array of the first byte of the item and repetition factor (number of identical items defined by one ITEM definition statement). If TYPE=ADDR is specified, 8 or 10 bytes of data are returned. This data contains a flag byte, the address of the array (if VS resident), the number of blocks defined for the array, and the size of the array (if unblocked) or the size of each block. Optionally, the number of items defined for the specified array(s) may also be retrieved.

GETARRAY EXAMPLE 1: This example will retrieve the content of array ABC into the area specified by the symbol ABCAREA. It is assumed that array ABC is less than or equal to 100 bytes.

```
       GETARRAY        NAME=ABC,DATA=ABCAREA,TYPE=DATA,...

ABCAREA DC XL100'0'
```

GETARRAY EXAMPLE 2: This example will retrieve the address and associated data for array number 1 into the area specified by symbol ADDR1.

```
       GETARRAY        NUMBER=1,DATA=ADDR1,TYPE=ADDR,...

ADDR1       DS   0F
            DC   XL1'0'       Flag byte
            DC   AL3(0)       Array address
            DC   H'0'         Number of blocks
            DC   H'0'         Size of array or block
```

GETARRAY EXAMPLE 3: This example will retrieve the address data for each array specified in list 'ADRL'. Since the increment (the second subparameter) of the FINDLST is greater than 10, the number of items in the array will be returned also. This increment causes the returned addresses to be moved into storage locations separated by 12 bytes for each entry.

```
       GETARRAY        NAMELST=ADDL,FINDLST=(FINDL,12),TYPE=ADDR

ADRL        DC   CL8'A1'
            DC   CL8'A2'
            DC   CL8'A3'
            DC   X'FF'        Flag byte to terminate the name list

FINDL       DS   0F
            DC   X'0'         Flag byte for array A1
            DC   AL3'(0)'     Address of array A1
            DC   H'0'         Number of blocks in array A1
            DC   H'0'         Length of array or each block
            DC   H'0'         Number of items in array 1
            DC   H'0'         Pad list to 12 bytes
            DC   2XL12'0'     Space for 2 additional lists as above for
                                 arrays A2 and A3
            DC   XL4'0'       Space for list termination flag
```

GETARRAY EXAMPLE 4: This example will cause the data from the arrays for which the addresses had been previously retrieved (as in example 3) to be retrieved. The data from the first array will be moved into area A1DATA; from the second array into area A2DATA, etc. It is assumed for this example that all three arrays are less than or equal to 100 bytes. For this example, it is assumed that the example 3 macro has been successfully executed to establish valid data into the following fields.

```
    GETARRAY    ADDRLST=(FINDL,12),DATALST=DATAL,TYPE=DATA,...

DATAL       DC    A(A1D,A2D,A3D)

A1D         DC    XL100'0'

A2D         DC    XL100'0'

A3D         DC    XL100'0'

FINDL       DS    0F
            DC    X'0'           Flag byte
            DC    AL3'0'         Addr. of array
            DC    H'0'           Number of blocks
            DC    H'0'           Length of block or array
            DC    H'0'           Number of items
            DC    H'0'           Unused
            DC    2XL12'0'       Space for 2 repeats of above
            DC    X'FF'          List terminator flag
```

PUTARRAY

The PUTARRAY macro is similar to the GETARRAY with the difference being that data is moved from the user's area to the VS resident data base. There is no TYPE= parameter on the PUTARRAY macro, so when compared to the GETARRAY macro, execution is always as if TYPE=DATA were specified.

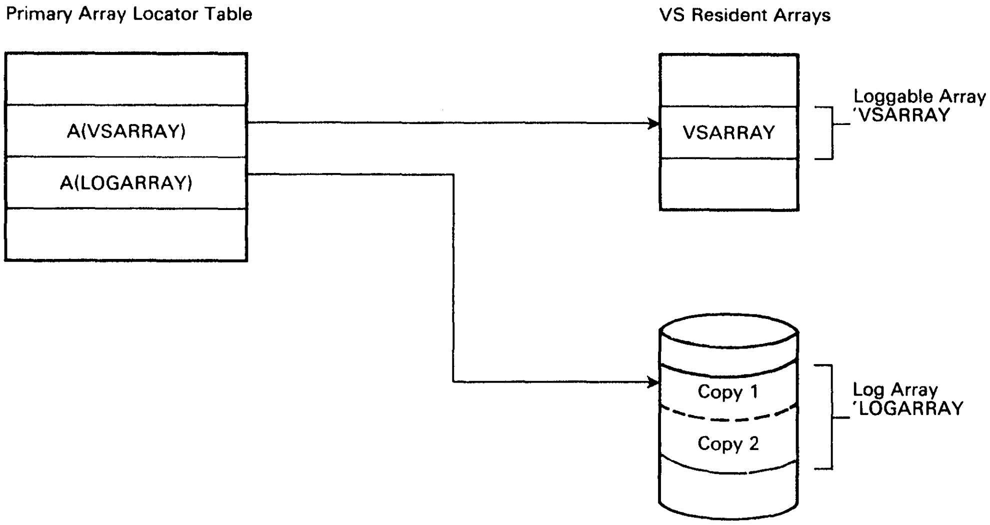
Data Base Logging

Data base logging is a Special Real Time Operating System option which may be selected at the Special Real Time Operating System SYSGEN time by the LOG macro.

During the offline utility phase, the user specifies which VS resident arrays are to be logged. These are called loggable arrays. A DA resident array with the array name specified by the LOGNAME keyword parameter in the ARRAY macro is constructed for each array to be logged. This array is called a log array. The LOGDD keyword parameter specifies the name of a data definition statement which describes a BDAM data set where the log array is to reside. The LOGCOPY keyword parameter specifies the number of historical copies that can be contained in this log array.

For example, the following ARRAY macro causes the offline utility routine to generate the following array structure.

```
   ARRAY        NAME=VSARRAY,LOGNAME=LOGARRAY,             *
                LOGDD=DBLOG1,LOGCOPY=2,...                 *
```



The first logging request for VSARRAY would cause the VS resident array to be copied into the space allocated for copy 1. The second logging request for VSARRAY would cause the VS resident array to be copied into the space allocated for copy 2. Since all the space allocated to the history files for VSARRAY has now been filled, the third logging request for VSARRAY would cause the VS resident array to be copied into the space allocated for copy 1 overlaying the data logged as a result of the first logging request.

To prevent a loss of history data, the user may specify the name of a user-written load module to be given control when the last block of the logging array has been filled through the LOGWRAP keyword parameter of the ARRAY macro. This load module will be entered via a PATCH to a dependent task. It is that load module's responsibility to preserve a record of the contents of the logged array at that time, possibly by dumping the log array to a sequential data set by the execution of a DUMPLOG macro call. If no user program has been specified, the user will not be notified that wraparound has occurred. The LOGFREQ keyword parameter consists of a code from 0 to 3 specifying the frequency at which the VS resident array is to be logged. A code of 0 indicates that it is to be logged only on demand, i.e., only when the user program executes a PUTLOG macro call. Codes 1 to 3 are used in conjunction with system generation parameters to specify the log frequency. A code of 1 is the highest frequency and 3 is the lowest. The Special Real Time Operating System logging routines will issue a PUTLOG for all VS arrays that are to be logged on the specified log frequency. Three macro calls; PUTLOG, GETLOG, and DUMPLOG, provide the user interface with the log subroutines.

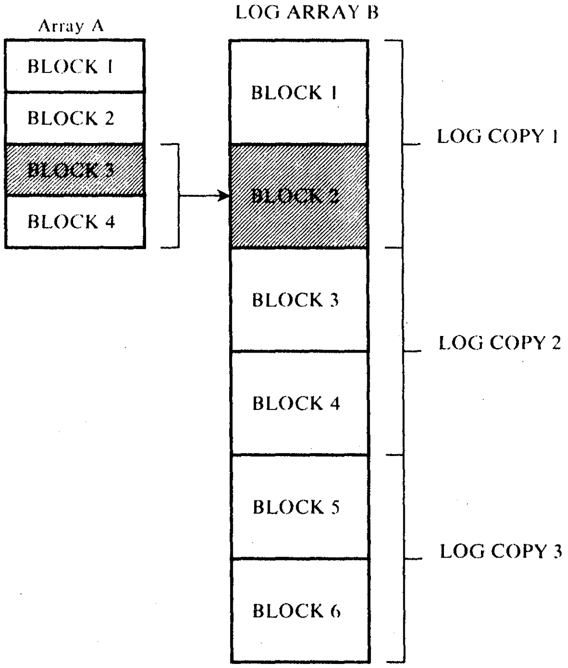
PUTLOG

The PUTLOG macro is used to copy the VS resident array to the proper copy of the log array. The NAME and NAMELST keyword parameters are used to specify the 8-character names of the VS resident array(s) from which data is to be logged. The NUMBER and NUMBLST keyword parameters are used to specify the 2-byte number(s) assigned to a numbered VS resident array(s) from which data is to be logged.

The user may replace a previously logged copy of the VS resident array without interrupting the normal sequential logging process. To accomplish this, the user would retrieve a log copy from the log array by executing a GETLOG macro call. This would read the requested log copy along with the log header into VS storage. The logheader contains the time this copy of VS resident array was logged and a pointer to its location in the log array. The user may then modify the data in this log copy and replace the log copy by executing a PUTLOG with the LOGHDR keyword parameter specifying the address of the previously read in logheader. The copy of the array that will replace the copy in the log array is assumed to immediately follow the specified logheader. If the logheader in the log array does not match the logheader indicated by the LOGHDR parameter, the logged copy will not be replaced. This will prevent the possibility of accidentally overlaying a newer log copy. The LOGHDR parameter is not valid with the NAMELST and NUMBLST keyword parameters.

The user also has the capability of updating selected blocks of the last logged copy in the log array for blocked VS resident arrays through the use of the PUTLOG with BLKLIST option. The BLKLIST keyword parameter identifies the blocks in the VS resident array that are to be logged. Each entry in the list must contain at least a 1-byte flag field and a 2-byte block number. A flag byte of X'40' indicates the last entry to be processed for a particular entry in the name list or number list. The PUTLOG when executed with the BLKLIST option will cause the log array block that corresponds to the specified VS resident array block to be updated in the last log copy of the log array. The entire log copy is not updated and repeating PUTLOG macro calls with the BLKLIST parameter will update the same log copy. A PUTLOG without the BLKLIST parameter will cause the entire VS resident array to be logged to a new log copy.

For example, assume that loggable array, A, consists of four logical blocks, and the associated log array, B, has been defined to contain three complete copies of loggable array A. Because of the physical block size of the data set that contains a log array, each copy of the loggable array may be placed in one or more blocks of the log array. Assume that each copy of loggable array A can be placed in two blocks of log array B. Therefore, the entire log array, B, would consist of six blocks (i.e., three copies and two blocks per copy).



The user might issue a PUTLOG to log an entire first copy of array A, and at sometime later issue a PUTBLOCK to update block 3 of the VS resident array A followed by a PUTLOG, with the BLKLIST option, using the same data list. The log block in the log array that contains the request loggable array block would be updated. That is, blocks 3 and 4 from the loggable array A would be moved into block 2 of the Log Array B.

GETLOG

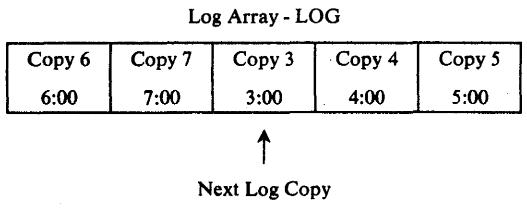
The GETLOG macro call can be used to retrieve copies of arrays that have been logged to the log array on the basis of time or by specifying a particular logheader. The NAME keyword parameter specifies the name of a VS resident array for which a logged copy is to be retrieved. The NUMBFR keyword parameter specifies the number of a VS resident numbered array for which a logged copy is to be retrieved. The AREA keyword parameter specifies the address of the user allocated area of storage where the logged copy of the array is to be written upon retrieval from the log data set. This area must be large enough to contain the entire log copy plus the logheader information.

The TIME keyword parameter specifies the time and day to be used as a comparison value to establish a relative starting point to determine which copy of the array will be retrieved from the log data set. An attempt will be made to locate a copy of the array logged at the exact time specified. If a copy of the array with the exact time cannot be found, the first copy of the array logged after that time will be used.

The LOGHDR keyword parameter specifies the address of an array logheader. Information in this logging header will establish a relative starting point to determine which copy of the array will be retrieved from the log data set. The logging header which was retrieved as part of a previous GETLOG macro call can be used to retrieve additional data by stepping either forward or backward in time. TIME and LOGHDR are mutually exclusive.

The STEP keyword parameter is used in conjunction with either the TIME or LOGHDR parameter to determine the copy of the VS resident array to be retrieved from the log array. The value specified in the STEP parameter is a signed number which may be either positive, negative, or zero. The absolute value of the number specified must be less than the number of log copies in the log array. The value indicates the number of copies prior to or after the log copy determined by either the TIME or LOGHDR parameter.

If the TIME, LOGHDR, and STEP parameters are omitted, then the latest logged copy of the array will be retrieved. For example, assume that the log array LOG contains five log copies of VS resident array, ARRAY.



Log Array - LOG

Copy 6	Copy 7	Copy 3	Copy 4	Copy 5
6:00	7:00	3:00	4:00	5:00

This array had been logged hourly for 7 hours starting at 1:00. Therefore, copies 1 and 2 would have been overlaid by copies 6 and 7, respectively, because of the wraparound processing. The following macro calls would all result in retrieving the same log copy, copy 4.

1. GETLOG TIME=T,STEP=1,...

```
 2. GETLOG TIME=X,STEP=-3,AREA=LH,...

 3. GETLOG LOGHDR=LH,STEP=0

T  DC '2:30' -- Actual value is in 10 millisecond units
X  DC '7:00'
LH DC 'COPY 4' -- Actual logheader.
```

Example 1 will find the first time logged after 2:30 and step 1 entry forward. Example 2 will find the first time logged after 7:00 and step backward 3 entries. Example 3 presumes that the logheader from example 2 exists in 'LH'; this example will retrieve the same data, since STEP=0.

DUMPLOG

The DUMPLOG macro call can be used to dump or unload the historical log copies of VS resident arrays from the log array to a user defined sequential data set. This sequential data set may then be accessed by user-written routines.

Note: Duplicate data set support is not provided for the user-defined sequential data set used in DUMPLOG processing.

The NAME and NAMELST keyword parameters specify the 8-character name(s) of the VS resident arrays for which the log array(s) are to be dumped. The NUMBER or NUMBLST keyword parameters specify the 2-byte number(s) assigned to a numbered array(s) for which the log array(s) are to be dumped.

The DUMPDD keyword parameter specifies the name of a data definition (DD) statement which describes a sequential data set to receive the dumped copies of the array from the log array. The USRDATA keyword parameter specifies the address of a 256-byte area of user data to be used as a dump header for each array on the sequential dump data set.

The log copies to be dumped are indicated by the STARTIM and STOPTIM keyword parameters. The STARTIM parameter specifies the time and day to be used to determine the first log copy to be dumped. An attempt will be made to locate a copy of the array with the exact time; if it cannot be found, the first copy of the array logged after that time will be used as the first log copy to be dumped. If STARTIM is omitted, dumping will commence with the oldest logged copy of the array.

The STOPTIM parameter specifies the time and day to be used to determine the last log copy to be dumped. An attempt will be made to locate a copy of the array logged at the exact time specified. If a copy of the array with its exact time cannot be found, the log copies of the array will be dumped until the most recently logged copy has been dumped or until the first copy of the array logged after that time has been dumped. If this parameter is omitted, dumping will terminate when the most recently logged copy of the array has been dumped.

Note that the DUMPLOG routine will insert a byte of 'FF' into the first byte of the logheader of the last copy of each array dumped to the sequential data set. This is done to indicate the end of the dump of each array to the user delog routine.

Data Base Refresh Function

The data base refresh function will allow the user to replace the current contents of one or more loggable VS arrays with the contents of the most recently logged copy(s) of the array(s). To invoke the function, the requesting program must PATCH the refresh program DPPDUPDL.

The PATCH request will consist of a list of arrays to be refreshed. If no list is specified, all refreshable arrays will be refreshed. A refreshable array is any array that was defined via the offline utility with the REINIT=YES parameter on the ARRAY macro. These modifications do not supersede the option of placing a DBREF card in the initialization stream if the data base is to be refreshed during initialization.

The PATCH macro format is as follows:

```
       Symbol      PATCH    TASK=taskname,EP=DPPDUPDL,
                            PARAM=(LIST),any other patch parameter
                            the user may want to specify

 LIST      DS          OH

           DC          CL8'name'

           DC          CL8'name'

           DC          H'number,'XL6'0'

           DC          H'number,'XL6'0'

           DC          X'FF

           or

           LA          R,LIST

 Symbol    PATCH       TASK=taskname,EP=DPPDUPDL,
                       PARAM=((R))

 R     is any register (2-12)

           or

 Symbol    PATCH       TASK=taskname,EP=DPPDUPDL
```

TASK= May be omitted to cause the program to execute as a dependent task or may specify any valid task name.

LIST Is the passed parameter list of the data base arrays to refresh. The list consists of 8-byte entries terminated by a byte of X'FF'. Each entry will consist of an 8-character array name or a half-word array number in 2 bytes followed by 6 bytes of zeros.

DATA RECORDING AND PLAYBACK

Data recording and playback provide a service which allows user programs to write data to a sequential data set and to retrieve that data at a later time. Both are standard Special Real Time Operating System services (not SYSGEN options). Data recording collects the data from several user programs, adds to it appropriate control information and user-supplied identifications, and writes the data to a sequential tape or disk data set. Data recording can be supressed or enabled through operator command (see "Input Message Processing"). Recorded data can later be selectively read back (based on time and ID) and passed to a user program or to the Special Real Time Operating System hexadecimal data print (hex dump) routine if no user program is supplied. The data may be printed, used to drive analysis programs, or used as test data to drive programs that are being developed. A 10-byte header is added to the data; otherwise, it is not changed in the recording playback sequence.

Both data recording and playback can be invoked in a single realtime job in one of two ways.

1. the two functions can use different data sets; that is, the playback can be from a data set that was recorded on a previous run and new data written on another data set.

2. the record function can be invoked, and the playback routine can be invoked for the same data set.

An example of the DRECOUT and DPBIN DD cards needed for the second case are as follows:

```
//DRECOUT DD DSN=username,DISP=(NEW,PASS),...
//DPBIN   DD DSN=*.DRECOUT,DISP=SHR,
//        VOL=REF=*.DRECOUT
```

Figure 2-11 shows the functions of data recording and playback.

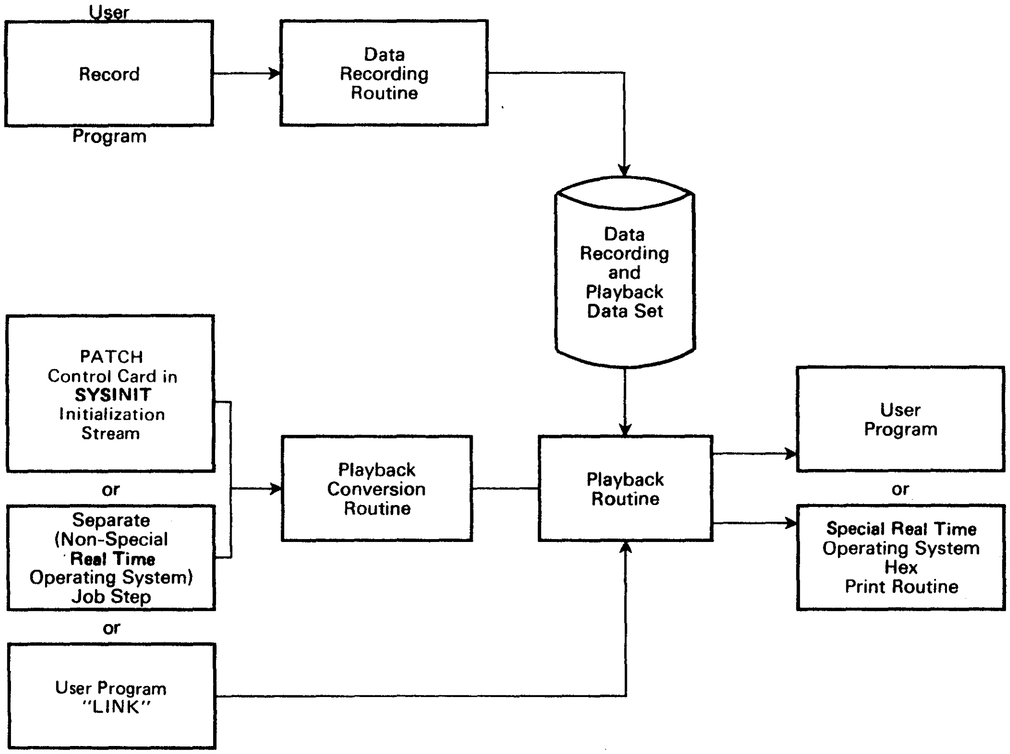

User
Record
Program
Data Recording Routine
Data Recording and Playback Data Set
PATCH Control Card in SYSINIT Initialization Stream
or
Separate (Non-Special Real Time Operating System) Job Step
or
User Program "LINK"
Playback Conversion Routine
Playback Routine
User Program
or
Special Real Time Operating System Hex Print Routine

Figure 2-11. Data recording and Playback Processing Overview

Data Recording Initialization

The data recording service is initialized at the Special Real Time Operating System initialization, so that any RECORD macro issued prior to the activation of data recording will be non-operational with a return code of 04. During realtime operation, the writing of data can be suppressed or enabled by the user. Data recording is enabled or disabled by an input message processing command.

```
'DREC  { ,ENABLE    { ,ALL }    [ id,id... ]  }
       {            { ,ADD }                  }
       {            { ,DEL }                  }
       { ,DISABLE                             }
```

DREC — Informs the input message processing routine that this reply is for data recording.

ENABLE/DISABLE — Causes data recording to be either enabled or disabled. Disable requires no other parameters.

ADD — Causes the following ID(s) to be placed in the Data Recording Table. Up to 20 IDs may be included in the table.

DEL — Causes the following ID(s) to be deleted from the Data Recording Table. DEL not followed by any ID causes all IDs to be deleted.

ALL — Causes all IDs to be enabled. No IDs are required.

id — A three-digit hexadecimal number (001-FFF) for which data is to be recorded.

Data Recording

Requests to record data for later playback are passed to the data recording function by the RECORD macro. With this macro, the user supplies an ID=(X'001-FFF'), the address (ADDR=) of the data, and the count (COUNT=) of bytes of data to be recorded (value of 1 to 65525). The data is written to a sequential data set defined by the user and is recorded on fixed length records. If the request is to record more data than will fit on one record, the data is split into two or more records to be reassembled into a single record when it is read back.

The data is time-tagged upon receipt (execution of the RECORD macro) and recorded in chronological order.

Data recording requests cannot span the partition boundary, so recording must be enabled in the partition where the program executing the RECORD macro resides. Recording may be enabled in both the MASTER and SLAVE partition simultaneously. When a given ID is enabled (either explicitly by entering that ID or implicitly with the ALL option) it is enabled for all programs in that partition. It is the responsibility of the user to select IDs that identify the source of the data and be meaningful when played back.

The following DD card is required by data recording:

//DRECOUT DD — defines a sequential data set to which the data will be written.

This data set will be opened (QSAM, LOCATE mode) when data recording is enabled and closed when data recording is disabled. Standard JCL conventions apply to this data set, and the user should be aware of the effect of all of the parameters that are specified. Some of the DD card parameters by which the user may affect data recording operation are as follows:

DISP= — If anything except MOD is specified, each time data recording is enabled, data will be written at the beginning of the data set. This may have the effect of over-writing data which was recorded by previous ENABLE/DISABLE sequences.

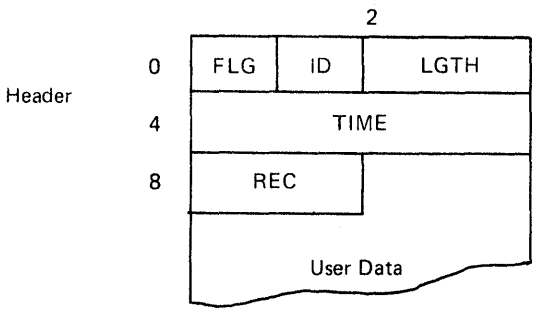
DCB=BLKSIZE= Defines the size of records written and QSAM buffers. The data is packed within the buffer by data recording. Specifying a large block size will reduce the number of I/O accesses but increase virtual storage use. A block size of less than 200 bytes is not recommended. If not specified, a block size of 2K bytes will be used; if specified, LRECL should be the same as BLKSIZE.

DCB=BUFNO= Specifies the number of buffers to be allocated by QSAM and, consequently, will affect the amount of waiting for I/O by the RECORD function. If not specified, three buffers will be allocated.

Data Playback

The data which has been recorded by the data recording facility may be read and passed to a user-supplied routine or to the Special Real Time Operating System hex data print (hex dump) routine based on time and IDs (which were assigned at data recording time).

The user specifies to the playback routine the data IDs and time range (start and stop times) for which data is to be processed. Also, the name of a user-supplied load module for data processing may be specified to the playback routine. If no user processing module is specified, the default processing routine is the Special Real Time Operating System hex data print routine. The user module may process the data according to the user's needs. The hex data print routine will supply a hex dump of the recorded data in a format similar to that of an ABEND dump. The data, when passed to a user load module, will be in the following format:



The header is a 10-byte field where FLG is four bits of flags set by data recording, ID is a 12-bit field that contains the identification supplied by the user, and LGTH is a 2-byte field which contains the length of the entry (including this 10-byte header). TIME is a 4-byte field that contains the time (in packed decimal format) that the data was recorded. User data is the data passed by the user. REC is data recording control data.

The playback routine may be invoked by any of three methods:

1. Through the Special Real Time Operating System initialization routine by PATCH control cards

2. As a separate (non-Special Real Time Operating System) job step

3. Through a LINK issued by a job running under the Special Real Time Operating System.

The following DD cards are required by data playback:

//DPBIN DD   Defines a sequential data set which contains data recorded by the RECORD macro.

//SRTODUMP DD Defines a sequential (printer) message data set.

Playback Via Patch Control Card

To invoke data playback at the Special Real Time Operating System subsystem initialization time through the use of a PATCH statement, the PATCH statement should be coded as shown:

```
[label]  PATCH   EP=DPPXPCON, [TASK=name,]  [QL=n,]
                 [ID=n,]
                 ┌       ┌JOBSTEP-n    ┐  ┐
                 │ PRTY= ┤             ├, │
                 └       └(taskname, n ┘  ┘
                 PARM=(C'STARTDATE' ,C'STARTIME',
                       C'STOPDATE',C'STOPTIME',
                       C'LM',C'COUNT',C'ID1',C'ID1A',
                       C'ID2',C'ID2A',C'ID3',C'ID3A',...)
```

See the section entitled "Special Real Time Operating System Initialization" for a complete description of the PATCH control statement. Only the parameters required by data playback are described here.

In some of the following parameter definitions, a zero has special meaning. In these cases, the parameter should be specified on the PATCH statement as a numeric value, using the F or X format (i.e., specified as F'0' rather than C'0').

DPPXPCON
Is the entry point of the playback conversion routine that converts the specified parameters to a form recognized by data playback and then passes the converted parameters via LINK to data playback.

STARTDATE
A date in the form of DD/MMM/YY (where DD is the day, MMM is the month (first three letters of the month are specified), YY is the year) specifies the day to start the playback process. Zero specifies that data playback is to start at the beginning of the data recording/playback data set. The characters 'ALL' specify that the entire data recording data set is to be played back. If ALL is specified, all other parameters are set to zero except the LM parameter.

STARTIME
Specifies the start time of data playback on the start date specified. Time is in the form of HHMMSST (where HH is hours, MM is minutes, SS is seconds, and T is tenth of seconds).

STOPDATE
A date, in the same format as STARTDATE, for which the last date is to be processed. Zero specifies that data recording is to stop at the end of the data recording/playback data set.

STOPTIME
Specifies the latest time on the date specified for which recorded data is to be processed. Time is in the same format as STARTIME.

LM
Is an 8-character entry point name of a load module to which data playback will pass the recorded data. If less than eight characters, it must be padded on the right with blanks. Zero specifies that the recorded data will be passed to the Special Real Time Operating System hexadecimal data print routine.

ID Count
Is the number of ID pairs (01-20) specified. The maximum number of ID pairs is 20.

IDn-IDm
Specifies a range of IDs to be played back within the time frame specified. IDn is the lowest ID in the range, and IDm is the highest ID in the range. If only one ID is to be played back, IDn and IDm must be identical. ID (001-FFF) is a three-digit hexadecimal number.

Example 1 shows three different patch cards for invoking data playback.

EXAMPLE 1:

```
//     EXEC          PGM=DPPINIT
//     -
//     -             DD cards required by the Special Real Time
       -             Operating System Initialization
//     -

//SYSINIT DD         *

P1      PATCH        EP=DPPXPCON,TASK=DPPXPCON,
                     QL=5,ID=7,PRTY=JOBSTEP-15,
                     PARAM=(C09/JAN/73',C'1520207',
                     C'09/FEB/73',C1730412',C'TESTMODE',
                     C'02',C'F20', C'F76', C'001',C'510')

P2      PATCH        EP=DPPXPCON,TASK=DPPXPCON,
                     PARAM=(X'0',C'1521459',X'0',
                     C'1643782',X'0',C'01',C'100',C'200')

P3      PATCH        EP=DPPXPCON,TASK=DPPXPCON,
                     QL=10,ID=9,PRTY=JOBSTEP-10,
                     PARAM=(C'ALL',X'0',X'0',X'0',
                         C'TESTMODE')
```

These three PATCH statements will cause data playback to be entered three times. PATCH statement P1 will cause any data recorded between 15 hours, 20 minutes, 20.7 seconds (3:20:20.7 pm) on January 9, 1973 and 17 hours, 30 minutes, 41.2 seconds (5:30:41.2 pm) on February 9, 1973 which has record IDs F20 through F76 or 001 through 510 to be passed to user load module TESTMODE.

PATCH statement P2 will cause all recorded data that has an ID 100 through 200 and was recorded between 15 hours, 31 minutes, 45.9 seconds and 16 hours, 43 minutes, and 78.2 seconds to be dumped to a SYSOUT data set by the Special Real Time Operating System raw data print routine. Because no dates are specified, the data set will be searched for the first data which has a time greater than the STARTIME, regardless of date and processed through the first data with a time greater than the STOPTIME regardless of date.

PATCH statement P3 will cause all data on the data set to be passed to load module TESTMODE. See the Special Real Time Operating System Initialization in Chapter 3, for a complete description of the PATCH cards.

Playback as a Separate Jobstep

When run as a separate (non-Special Real Time Operating System) job step, either in a background partition or on an offline CPU, the parameters are passed to the data playback non-realtime initialization through the PARM parameter of the JCL EXEC statement.

```
//stepname     EXEC    PGM=DPPXNRTI,
                           PARM='STARTDATE,STARTIME,
                               STOPDATE,STOPTIME,
                               LM,COUNT,ID1,
                                   ID1A,ID2,ID2A,
                                   ID3,ID3A,...'
```

stepname
Is the name of the job step.

DPPXNRTI
Is the name of the non-realtime Special Real Time Operating System program to which the parameters will be passed.

STARTDATE, STARTTIME, STOPDATE, STOPTIME, LM, COUNT, ID
Have the same meaning as described for PATCH control statement.

Note: Every playback parameter must be specified except when ALL is specified.

When ALL is passed to the non-realtime playback routine (DPPXNRTI) with a load module name, the parameters should be in the following format:

```
//stepname EXEC PGM=DPPXNRTI,PARM='ALL bbbbbb, LM'
```

where ALL is followed by six blanks as the first parameter and the load module name as the second parameter.

The fields within the PARM string are positional, and each field must occupy the exact number of positions allocated to that field as follows:

STARTDATE	9	
STARTIME	7	
STOPDATE	9	
STOPTIME	7	
LM	8	If a Load Module name is specified or 1 if zero is specified
COUNT	2	
ID	3 each	

All fields must be separated by commas.

In examples 2 and 3 the Special Real Time Operating System playback is run as a separate (Non-Special Real Time Operating System) job step.

EXAMPLE 2:

```
 // EXEC  PGM=DPPXNRTI,
    PARM='07/JAN/73,0800000,07/FEB/73,
     0900000,0,02,020,025,040,050'
```

All data that has an ID in the range 020 through 025 and 040 through 050 and that was recorded after 08:00:00.0 on January 7, 1973 and 09:00:00.0 on February 7, 1973 will be printed by the Special Real Time Operating System raw data print routine.

EXAMPLE 3:

```
 // EXEC PGM=DPPXNRTI,
   PARM='ALLbbbbbb,TESTMODE'
```

All data on the data set will be passed to load module TESTMODE.

Playback Via Link

The LINK macro instruction may be used to invoke data playback. The LINK macro should be in the following format:

```
 EXAMPLE        CSECT
                instructions

 symbol         LINK     EP=DPPXDPB,PARAM=(PARM)
                or
                LA       R,PARM
                LINK     EP=DPPXDPB,PARAM=((R))
                         instructions
 PARM           DS       0F
 STARTDAT       DS       CL9
 STARTTIM       DS       PL4
 STOPDATE       DS       CL9
 STOPTIME       DS       PL4
 LM             DS       CL8
 IDCOUNT        DS       AL2
 ID1            DS       XL2
 ID1A           DS       XL2
 ID2            DS       XL2
 ID2            DS       XL2
 ID2A           DS       XL2
 ID3            DS       XL2
 ID3A           DS       XL2
```

R
Is a general purpose register.

DPPXDPB
Is the data playback entry point name.

PARM
Is the address of the playback parameters.

The playback parameters for the LINK should be in the following format:

Bytes	Field Name	Field Description, Contents, Meaning
9	STARTDAT	
4	STARTTIM	
9	STOPDAT	
4	STOPTIME	
8	LM	
2	IDCOUNT	
2	ID	
2	ID	
	.	
	.	additional IDs in pairs
	.	

Examples 4 and 5 show a LINK to the playback function from a user coded program.

EXAMPLE 4:

```
 EXAMPLE4       CSECT
                    instructions
                LINK EP=DPPXDPB,PARAM=(PARM)
                .
                .
                .
                DS  OF
 PARM           DC  CL9'09/JAN/73'
                DC  PL4'1540071'
                DC  CL9'09/FEB/73'
                DC  PL4'1650509'
                DC  CL8'TESTMODE'
                DC  AL2(3)
                DC  XL2'111',XL2'222'
                DC  XL2'100;XL2'110'
                DC  XL2'FF0',XL2'FFF'
                END
```

A job running under the Special Real Time Operating System will LINK to the Special Real Time Operating System data playback routine. All data that has an ID in the range 111 through 222, 100 through 110, and FF0 through FFF and that was recorded between 15 hours, 40 minutes, 07.1 seconds and 16 hours, 50 minutes, 50.9 seconds on 09/JAN/73 will be passed to load module TESTMODE.

EXAMPLE 5:

```
 EXAMPLE5  CSECT
               instructions
           LA 1,PARM
           LINK EP=DPPXDPB,PARAM=((1))
           .
           .
           .
 PARM      DS  OF
           DC  CL9'ALL'
           DC  PL4'0'
           DC  XL9'0'
           DC  PL4'0'
           DC  CL8'TESTMODE'
```

A job running under the Special Real Time Operating System will LINK to the Special Real Time Operating System data playback routine. All data in the data set will be passed to load module TESTMODE.

HIGH-LEVEL LANGUAGE INTERFACES

The Special Real Time Operating System routines provide an interface to allow PL/I and FORTRAN users to use most of the services provided by the Special Real Time Operating System. The interface routines are independent of the compiler level or the optimizing compilers. Figure 2-12 lists the Special Real Time Operating System macros supported by the interface routines for PL/I. The macros in the figure are also supported for FORTRAN, but there are no default structures.

Macro Name	ID	PL/I	
		Structure Name	Member Name
PATCH	0	PATCHSTR	PATCHDEF
PATCH Param	0	PARMSTR	PARMDEF
PTIME	4	PTIMESTR	PTIMEDEF
PTIME	4	PTIMRSTR	PTIMRDEF
DPATCH	8	DPACHSTR	DPACHDEF
REPATCH	12	REPCHSTR	REPCHDEF
GETARRAY	16	ARRAYSTR	ARRAYDEF
GETITEM	20	ITEMSTR	ITEMDEF
GETBLOCK	24	BLOCKSTR	BLOCKDEF
PUTARRAY	16	ARRAYSTR	ARRAYDEF
PUTITEM	20	ITEMSTR	ITEMDEF
PUTBLOCK	24	BLOCKSTR	BLOCKDEF
MESSAGE	40	MESAGSTR	MESAGDEF
PUTLOG	44	PTLOGSTR	PTLOGDEF
GETLOG	48	GTLOGSTR	GTLOGDEF
DUMPLOG	52	DPLOGSTR	DPLOGDEF
RECORD	56	RECRDSTR	RECRDDEF
PATCH WAIT	60	WAITSTR	WAITDEF

Figure 2-12. Macros Supported by FORTRAN-PL/I Interface Routines

All interface routines are invoked as shown in Figure 2-13. The parameters are passed using standard linkage conventions to the assembler language interface routine. The interface routine adjusts the parameter list and then issues an execute form of the appropriate macro to invoke the desired service. After the service routine has completed execution, the interface routine stores the return code for use by the calling program and returns to the caller.

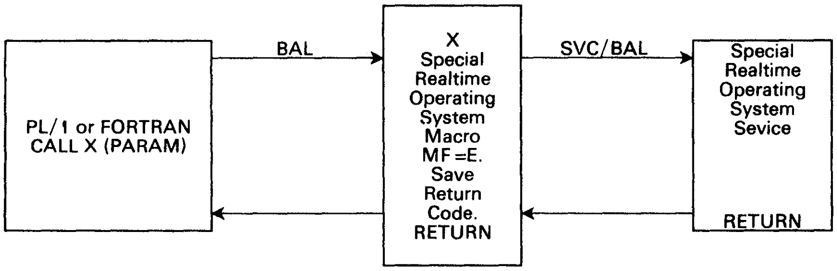


Figure 2-13. High-Level Language Interfaces for the Special Real Time Operating System Services

The high level language user must refer to the Special Real Time Operating System macros section when using the language interfaces, as more details are given with each macro description.

The Special Real Time Operating System-PL/I Interface

The PL/I interfaces to the Special Real Time Operating System services are designed to be independent of the PL/I compiler used. This means "dope vectors" or "locator/descriptors" are not referenced by the interface routines. To avoid referencing "secondary" pointers, the parameter of a CALL statement must point to the first element of the structure defining the parameter list.

```
        DCL 1 PATCHSTR,
            2 MACID,
            2 RC,
            .
            .
            .
```

For example, given the above structure, the call statement would have to be CALL DPPPIF(PATCHSTR.MACID) for the correct parameter list to be passed to the interface routine DPPPIF.

All Special Real Time Operating System services invoked by a PL/I program have unique parameter lists which can be described by a structure. An aid to the PL/I programmer are default structure definitions. The programmer may invoke them through the compiler preprocessor option - %INCLUDE. A list of the PL/I structure definitions and names is included in Figure 2-12. Each of the default structures is explained in the following sections describing the Special Real Time Operating System services provided for PL/I programs. Any option changes made by the PL/I program to a default structure must be reset if the structure is reused and the option is not desired.

In addition, users of the default structure will notice the two fields (MACID and RC) at the beginning of each. They are common to every structure used as a parameter when calling DPPPIF. MACID is initialized in the default structures with the correct value to tell the interface routine which service is being requested. RC is where the return code from the service routine is stored.

PL/I programs in a normal OS/VS1 job shop environment are initiated, the PL/I Prolog routines and the user program are executed, and at termination the PL/I Epilog routine is executed. In a realtime environment where the PL/I program is to be cyclically executed, the PL/I Interface routines provide facilities to allow the PL/I program to keep its resources across cyclic executions and to execute cyclically without incurring the overhead of Prolog and Epilog for each execution following the initial execution. This facility applies only to independent tasks that are PATCHed with the EP= parameter specifying the same EP name. Figure 2-14 shows the coding of a PL/I program using this facility.

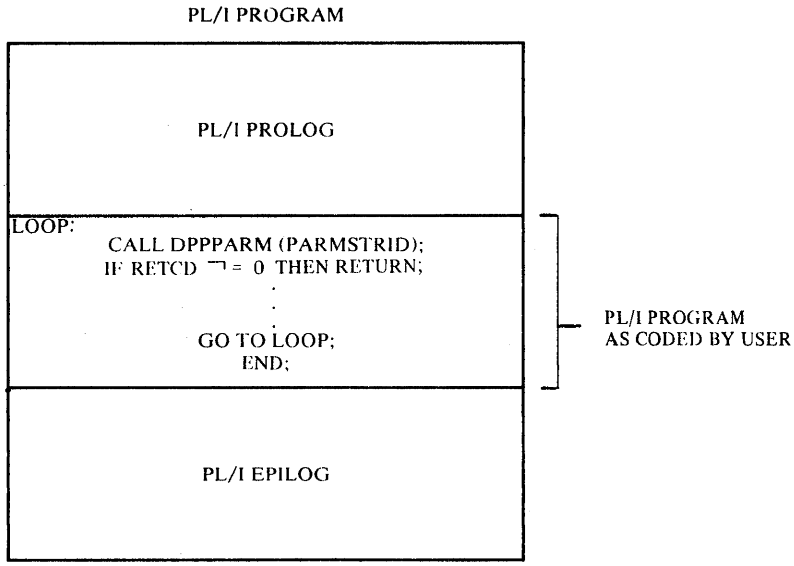


Figure 2-14. PL/I Example

The following is a series of PATCHes to PL/I programs which will illustrate when a program would be forced through Epilog.

```
        A       PATCH       TASK=A,EP=PLIPROG
        B       PATCH       TASK=A,EP=PLIPROG
        C       PATCH       TASK=A,EP=PLIPROG
        D       PATCH       TASK=A,EP=PLIEXMP
        E       PATCH       TASK=A,EP=PLIPROG
        F       PATCH       TASK=A,EP=PLIEXMP
        G       PATCH       TASK=A,EP=PLIEXMP
```

where: PLIPROG and PLIEXMP are PL/I programs coded as shown in the previous example.

PATCH A executes Prolog for PLIPROG, then the body of PLIPROG. When the body finishes, a second CALL is made to DPPPARM. PATCH 3 then executes without going through Prolog. PATCH B in turn finishes and again calls DPPPARM. PATCH C then executes - again without going through Prolog. When PATCH C finishes, another call is made to DPPPARM. The PL/I interface routine determines that the next PATCH (D) is to a different program. A non-zero return code forces PATCH C to terminate and thus execute PL/I Epilog. PATCH D then executes, going through PROLOG and the code body for PLIEXMP. PATCH D finishes and again calls DPPPARM. Once again, the interface recognizes that the next program to be executed is different and returns a non-zero return code. Program DPPEXMP is forced through Epilog. PATCH E passes through both Prolog and Epilog and PATCH F passes through Prolog and PATCH G executes without Prolog. Then, on the next call to DPPPARM, Task A is placed in a wait state until another PATCH to it is received.

PATCH-to-PL/I Interface

PL/I programs cannot easily retrieve parameters passed via register 1. To obtain the parameters in a PL/I program invoked by PATCH, an interface routine DPPPARM and a structure PARMSTR, which may be copied into the PL/I program by %INCLUDE PARMDEF; are provided. The following PL/I statements define PARMSTR:

```
      DCL 1 PARMSTR,
          2 ID FIXED BIN INIT(0) ,          /* RESERVED */
          2 RETCD FIXED BIN INIT(1),        /* 0 IF PARMS CHANGED */
          2 XCVT POINTER,                   /* A(XCVT) */
          2 RESOURCE POINTER,               /* A(RESOURCE TABLE) */
          2 PARMS POINTER;                  /* A(PATCH PARAMETERS) */
```

PARMSTR
Is the name of the structure used to obtain the PATCH problem parameters.

ID
Is reserved halfword initialized to zero.

RETCD
Is a halfword binary number indicating the validity of the pointer value in PARMS. If not zero, the PL/I program should not use the address in PARMS and should return control to the system. If zero, PARMS contains a valid address.

XCVT
Specifies the address of the Special Real Time Operating System control block XCVT.

RESOURCE
Specifies the address of a two fullword area available to all programs executing under the current task.

PARMS
Specifies the address of the problem parameters being passed by a PATCH to the program.

The PL/I program using this interface must declare the structure only once and in the highest block. The structure must be reused without reinitializing. If the program CALLs for another set of PATCH parameters and the task work queue is empty, the program will be placed in a wait until a PATCH is issued for the task.

The example below is the proper method for using the structure. This example uses the default structure PARMSTR to obtain the PATCH pointers. The structure defining the parameter list is based on the PARMS pointer variable. Note that the PL/I program loops back to the CALL statement and that the only exit occurs if the return code from DPPPARM is not zero. This minimizes the execution of PL/I Prolog and Epilog.

```
DCL 1 PARMSTR,
    2 ID FIXED BIN INIT(0),
    2 RETCD FIXED BIN INIT(0),
    2 XCVT POINTER,
    2 RESOURCE POINTER,
    2 PARMS POINTER;

DCL 1 PARAMETER BASED (PARMS),
    2 LENG FIXED BIN,
    2 PATCHID FIXED BIN;

LOOP:
    CALL DPPPARM (PARMSTR.ID);
    IF RETCD = 0 THEN RETURN;
        •
        •
        •
    normal execution
        •
        •
        •
    GOTO LOOP;
    END program;
```

PL/I-PATCH Interface

The default structure which defines the parameter list for invoking the PATCH service may be copied into the program by %INCLUDE PATCHDEF. The PL/I statements and definitions are listed as follows:

```
DCL   1 PATCHSTR,                           /* PATCH STRUCTURE */
      2 MACID FIXED BIN INIT(0),            /* PATCH MACRO ID */
      2 RC FIXED BIN INIT(0),               /* RETURN CODE */
      2 PACHPARM POINTER,                   /* A(PARAMETERS) */
      2 TASKNAME CHAR(8) INIT(' ')          /* TASKNAME */
      2 EPNAME CHAR(8) INIT('IEFBR14'),  /* LOAD MODULE */
      2 NAME CHAR(8) INIT{' ')              /* RELATIVE TASK OF VALUE */
      2 QUEUE FIXED BIN INIT(1),            /* DEFAULT = 1 */
      2 VALUE FIXED BIN INIT{0),            /* DEFAULT = 0 */
      2 ECB POINTER,                        /* ECB ADDRESS */
      2 FREEL FIXED BIN(31,0) INIT(0),   /* RESERVED */
      2 FREEA FIXED BIN(31,0) INIT(0),   /* RESERVED */
      2 TCBX FIXED BIN(31,0) INIT(0) ,   /* TCB EXTENSION */
      2 PFLAGS                              /* FLAG OPTIONS IF BIT IS SET ON */
         3 (FO,                             /* RESERVED */
         MASTER,                            /* PATCH MASTER PARTITION */
         SLAVE,                             /* PATCH SLAVE PARTITION */
         F3,                                /* RESERVED */
         REPCH,                             /* ECB REPATCH */
         QPOS,                              /* QPOS=FIRST */
         DPCH,                              /* QPOS=DPATCH */
         DEL) BIT(1) INIT('0'B);            /* EP DELETE */
```

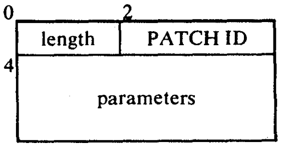
PATCHSTR
The name of the default structure.

MACID
Specifies the halfword binary value set to zero to identify the PATCH service request.

RC
Specifies a halfword binary field containing the return code from the service routine. The return codes are described in the PATCH macro definition.

PACHPARM
Specifies the address of a parameter list being passed. The format is a halfword binary value (minimum value is 4) describing the length of the entire parameter list, followed by a halfword binary value from 0 to 255 called the PATCH ID with the remainder of the list being the parameters. The diagram below represents the format of a PATCH parameter list.

Note: If the list is greater than 8 bytes, the interface routine will move it to a GETMAIN area to be freed when processing of the work queue is completed.



TASKNAME
Specifies a 1 to 8 character name which is the name of the task being referenced by this PATCH. If the task does not exist, one by that name will be created.

EPNAME
Specifies a 1 to 8 character valid program name which is the name of the program to be scheduled under the task being created with the PATCH.

NAME and VALUE

Specifies a task name and a value which will determine the priority of the new task. VALUE will be subtracted from the dispatching priority of the specified task. VALUE may range from 0 to 255 with zero default. See PRTY option of PATCH macro for further detail.

QUEUE

Specifies the number of work queue entries to be provided for the new independent task. Any decimal value from 0 to 255 may be specified. The default value is 1. A work queue entry provides space to queue PATCHes which have not been executed by the task. If 0 is specified as the queue length, the task accepts one PATCH, works on that request, and when completed, waits for the next request. If a PATCH is issued for that task while the task is busy, it is not executed. If the queue length is 1, the task can accept one PATCH even while it is busy. Any PATCH parameters waiting in the queue when a task completes processing the current request will be executed one at a time, with the top of the queue executed next. This procedure is the same for all queue values from 0 to 255.

ECB

Specifies the address of the ECB within a WAITSTR which is to be used in a CALL DPPPIF. This ECB is posted when processing for this PATCH is completed. The REPCH flag causes the ECB to be posted with the address to be used in the REPATCH macro if this PATCH is not executed because of a DPATCH or a QPOS=FIRST PATCH with the queue full. Default is no ECB. See PL/I PATCH WAIT.

FREEL and FREEA

Are reserved.

TCBX

Specifies the address of the TCB extension control block (TCBX) for an existing independent task. The TCBX address is returned in structure after each PATCH. Use of this operand with all PATCHes to the same task after the initial PATCH will reduce system processing time. Note that other parameters must still be specified for verification or in the event the task has been DPATCHed.

PFLAGS

Are PATCH option flags as described below:

F0 and F3

Are reserved.

MASTER

Specifies this is a PATCH to the MASTER partition.

SLAVE

Specifies this is a PATCH to the SLAVE partition.

REPCH

Specifies that the ECB will be posted when a REPATCH control block is built. Default is no REPATCH control block.

QPOS and DPCH

Specifies in the task work queue where this work request is to go if the task is busy. If QPOS is on, the request is to be placed so as to be processed before those already on the queue. If DPCH is on, the processing for this PATCH will not be executed until a DPATCH is issued for this task. Default is last on the work queue.

DEL

Specifies that a DELETE is issued for the EP name after processing completes for this PATCH. Default is no.

The Special Real Time Operating System PATCH service may be invoked by including the PATCHDEF in the PL/I program, completing the required information within the structure including building a parameter list and calling the interface routine DPPPIF with the PATCHSTR. Examples of using the PATCH facility follow.

In Example 1, structures are declared for a parameter list and the PATCH structure. The task DPPZTS00 is created with a queue length of 1. Program DPPZTS13 is executed, and the parameter list contains only the length field and a PATCH ID of 10. The new task must have the same priority as the task issuing the PATCH. The PATCHing program does not want notification of the completion of the PATCH. Note that if the task already exists, the PFLAGS indicate this work request will be queued behind any others on the queue.

```
 DCL    1 PARAMETER,
        2 LENG FIXED BIN,
        2 PATCHID FIXED BIN,
        2 PARAMS (10) FIXED BIN (31,0);

 DCL    1 WAITSTR,
        2 MACID FIXED BIN INIT (60),
        2 RC FIXED BIN INIT (0),
        2 ECBX FIXED BIN (31,0) INIT (0);

 %INCLUDE PATCHDEF;

 DCL    1 PATCHSTR,
        2 MACID FIXED BIN INIT (0),       /* PATCH MACRO ID */
        2 RC FIXED BIN INIT (0),          /* RETURN CODE */
        2 PACHPARM POINTER,               /* A(PARAMETERS) */
        2 TASKNAME CHAR(8) INIT (' ')     /* TASK NAME */
        2 EPNAME CHAR(8) INIT ('IEFBR14') /* LOAD MODULE */
        2 NAME CHAR(8) INIT(' '),         /* RELATIVE TASK OF VALUE */
        2 QUEUE FIXED BIN INIT(1),        /* DEFAULT = 1 */
        2 VALUE FIXED BIN INIT(0),        /* DEFAULT = 0 */
        2 ECB POINTER,                    /* ECB ADDRESS */
        2 FREEL FIXED BIN(31,0) INIT(0),  /* RESERVED */
        2 FREEA FIXED BIN(31,0) INIT(0),  /* RESERVED */
        2 TCBX FIXED BIN(31,0) INIT(0),   /* TCB EXTENSION */
        2 PFLAGS.                         /* FLAG OPTIONS IF BIT IS SET ON */
           3 (F0,                         /* RESERVED */
           MASTER,                        /* PARTITION=MASTER */
           SLAVE,                         /* PARTITION=SLAVE */
           F3,                            /* RESERVED */
           REPCH,                         /* ECB REPATCH */
           QPOS,                          /* QPOS=FIRST */
           DPCH,                          /* QPOS=DPATCH */
           DEL) BIT(1) INIT('0'B);        /* EP DELETE */
        LENG = 4;
        PATCHID = 10;
        PACHPARM = ADDR(PARAMETER.LENG);
        TASKNAME = 'DPPZTS00';
        EPNAME = 'DPPZTS13';
        CALL DPPPIF (PATCHSTR.MACID);
           .
           .
           .
```

Example 1

In Example 2, assume that the CALL in Example 1 has returned, and a dependent task is to be created at a priority of 10 less than the task DPPZTS00 and that program DEPENDX is to be passed a parameter list of 10 numbers with a PATCH ID of 2. The PATCHing program will wait for the dependent task to complete. The WAIT function is done via a CALL to the interface routine using the WAITSTR structure.

```
 DCL    1 PARAMETER,
        2 LENG FIXED BIN,
        2 PATCHID FIXED BIN,
        2 PARAMS (10) FIXED BIN (31,0) ;

 DCL    1 WAITSTR,
        2 MACID FIXED BIN INIT(60) ,
        2 RC FIXED BIN INIT(0) ,
        2 ECBX FIXED BIN (31,0) INIT(0);

 %INCLUDE PATCHDEF;

 DCL   1 PATCHSTR,
        2 MACID FIXED BIN INIT(0),       /* PATCH MACRO ID */
        2 RC FIXED BIN INIT(0),          /* RETURN CODE */
        2 PACHPARM POINTER,              /* A(PARAMETERS) */
        2 TASKNAME CHAR(8) INIT(' '),    /* TASK NAME */
        2 EPNAME CHAR(8) INIT ('IEFBR14'), /* LOAD MODULE */
        2 NAME CHAR(8) INIT(' ')         /* RELATIVE TASK OF VALUE */
        2 QUEUE FIXED BIN INIT(1),       /* DEFAULT = 1 */
        2 VALUE FIXED BIN INIT(0),       /* DEFAULT = 0 */
        2 ECB POINTER,                   /* ECB ADDRESS */
        2 FREEL FIXED BIN(31,0) INIT(0), /* RESERVED */
        2 FREEA FIXED BIN(31,0) INIT(0), /* RESERVED */
        2 TCBX FIXED BIN(31,0) INIT(0) , /* TCB EXTENSION */
        2 PFLAGS,                        /* FLAG OPTIONS IF BIT IS SET ON */
          3 (FO,                         /* RESERVED */
          MASTER,                        /* PARTITION=MASTER */
          SLAVE,                         /* PARTITION=SLAVE */
          F3,                            /* RESERVED */
          REPCH,                         /* ECB REPATCH */
          QPOS,                          /* EPOS=FIRST */
          DPCH,                          /* QPOS=DPATCH */
          DEL) BIT(1) INIT('0'B) ;       /* EP DELETE */
                  .
                  .
                  .
          CALL DPPPIF (PATCHSTR.MACID);/*EXAMPLE 1*/
          LENG = 44;
          PATCHID = 2;
          TASKNAME = ";
          EPNAME = 'DEPENDX';
          NAME = 'DPPZTS00';
          VALUE = 10;
          ECB = ADDR(ECBX);
          CALL DPPPIF (PATCHSTR.MACID);

          IF PATCHSTR.RC <8 THEN DO;
              CALL DPPPIF (WAITSTR.MACID);
          END;
               .
               .
               .
```

Example 2

PL/I-PTIME Interface

The Special Real Time Operating System PTIME service provides two different functions, time and PATCH, issued on a time queue basis. Therefore, two default structures may be copied into the program by

%INCLUDE PTIMEDEF and PTIMRDEF which define the parameter lists for the PTIME services. The PL/I statements and their meanings are as follows:

```
DCL   1 PTIMRSTR,                       /* STRUCTURE FOR PTIME TYPE=RET */
      2 MACID FIXED BIN INIT(4),        /* PTIME SERVICE */
      2 RC FIXED BIN INIT(0),           /* RETURN CODE */
      2 TYPE FIXED BIN(31,0) INIT(0),   /* PTIME CALL TYPE */
      2 TIME FIXED BIN(31,0) INIT(0),   /* CURRENT TIME */
      2 TIMDSECT POINTER;               /* A(TIME ARRAY) */
```

PTIMRSTR
Is the name of the default structure used to obtain the current time and the address of the time array.

MACID
Is the halfword binary value set to 4 to identify a PTIME service request.

RC
Is a halfword binary value containing the return code from the service request. always 0.

TYPE
Is a fullword binary number identifying the PTIME service being requested. For this structure, it is For this structure, it is always 0.

TIME
Is a fullword binary field which will contain the current time of day in 10 millisecond units when the interface routine returns.

TIMDSECT
Specifies the address of the Special Real Time Operating System time array when the interface routine returns.

```
DCL   1 PTIMESTR,                           /*PTIME STRUCTURE FOR ADD,MOD,DEL */
      2   MACID FIXED BIN INIT (4),         /* PTIME SERVICE */
      2   RC FIXED BIN INIT (0),            /* RETURN CODE */
      2   TYPE FIXED BIN (31,0) INIT(4),    /* PTIME CALL TYPE */
      2   STIME FIXED BIN (31,0) INIT(0),   /*START TIME */
      2   ITIME FIXED BIN (31,0) INIT(0),   /*INTERVAL TIME */
      2   ETIME FIXED BIN (31,0) INIT(0),   /*STOP TIME */
      2   PATCH POINTER,                    /*A(PATCH SUPL)*/
          2   PARMS POINTER,                /*A(PARAMETERS)*/
          2   START,                        /*FLAGS DEFINE STIME CONTENTS */
              3 (F0,F1,F2,F3,F4,            /*RELATIVE TIME */
                   SADJFLAG,                /*ADJUSTED TIME */
                   STODFLAG,                /*TIME OF DAY */
              SRELFLAG) BIT (1) INIT ('0'B), /*RELATIVE TIME */
          2   PURGE,                        /*FLAGS DEFINE PTIME PURGE OPTIONS */
              3 (F0,F1,F2,F3,               /*RESERVED */
                   PURGEI,                  /*DPATCH = I */
                  PURGEW,                   /*DPATCH = W */
                  PURGEC,                   /*DPATCH = C */
                  PURGEU) BIT (1) INIT ('0'B), /*DPATCH = U */
              2   STOP,                     /*FLAGS DEFINE ETIME CONTENTS */
              3 (F0,F1,F2,F3,               /*RESERVED */
                   ECNTFLAG,                /*COUNT VALUE */
                   EADJFLAG,                /*ADJUSTED TIME */
                   ETODFLAG,                /*TIME OF DAY */
              ERELFLAG) BIT (1) INIT ('0'B); /*RELATIVE TIME */
```

PTIMESTR
Is the name of the default structure used to create or modify PATCH

service requests by time queue.

MACID
Is a halfword binary value set to 4 to identify a PTIME service request.

RC
Is a halfword binary value containing the return code from the service request. If the return code is 8 or larger, the PTIME was not successful, and the existing PTIME specification was not changed. The return codes are defined in the macro description.

TYPE
Is a fullword binary number specifying the type of PTIME service requested. Values may be 4, 8, or 12. If 4, a PTIME queue element (PTQE) is created which controls the PATCHes issued according to the PTIME request. Since the PTQE exists independently of the creating task and may be modified (8) or deleted (12), the PTQE is referred to by task name, entry point name, and the PATCH ID value in the passed parameter list. Either task name or entry point name must be given for a modify (8) or delete (12) request. However, if only a task name or entry point name is specified, all PTQEs with that name are deleted or modified. The default is to create a PTQE (4).

STIME*
Is a fullword binary number specifying the time in 10 millisecond units of the first PATCH. The flags START specify the value in this field.

SRELFLAG
If on, the first PATCH will be issued at current time plus the value of STIME.

STODFLAG
If on, the first PATCH will be issued when current time equals the value of STIME. If STIME is less than current time, the PATCH will occur the next day.

SADJFLAG
If on, the time of the first PATCH is calculated by assuming STIME contains the time of day (TOD), except that the value in ITIME is added to STIME until that value is greater than current time.

ITIME*
Is a fullword binary number specifying the interval in 10 millisecond units between successive PATCHes.

ETIME*
Is a fullword binary number specifying when the PTQE is to be deleted. The flags STOP identify the value in this field.

*All time values are in 10 millisecond units and must not exceed 24 hours.

ECNTFLAG
If on, ETIME contains a count of the number of PATCHes to be issued by this PTQE.

ERELFLAG*
If on, ETIME contains a time value in 10 millisecond units, when added to the current time equals the stop time.

ETODFLAG*
If on, ETIME contains the stop time ın 10 millisecond units.

EADJFLAG*
If on, the stop time is calculated by assuming ETIME contains the time of day (TOD) in 10 millisecond units, except that the value in ITIME is added to ETIME until the value is greater than current time.

*Regardless of what value is calculated for a stop time, if it is less than the calculated start time (see STIME above), a 24-hour value is added to the stop time until the stop time exceeds the start time.

Note: If all the STOP flags are zero and ETIME is zero, the PTIME is assumed to be infinite, and PATCHes will be issued until a PTIME to modify (8) or delete (12) is issued for that task and/or entry point name.

PATCH
Is the address of the supervisor portion of the PATCH parameters. The options provided will be used by PTIME to issue PATCHes based on the above time options. If PATCHSTR (the default structure) is used, this parameter must point to TASKNAME. All information desired for the PATCH by PTIME must be supplied prior to CALLing the interface routine.

RESTRICTION: Queue Position of DPATCH is not permitted (PFLAGS.DPCH set to 1).

PARMS
Is the address of a parameter list to be passed by the PATCH issued by PTIME. See PL/I PATCH Interface for format. Note that if this parameter list is greater than 8 bytes, the interface routine will move it to a GETMAIN area to be freed when the PTQE is destroyed.

START
Specifies the start time option flags which define the contents of STIME. Only one of the flags must be set. See STIME for flag definitions.

PURGE
Is the flag that controls the kind of DPATCH which will be issued when the PTQE is destroyed. If no flag is set, no DPATCH is issued. Flags at a PTIME delete (12) will override the flags when the PTQE was created (4) or modified (8) last. Only one flag may be set.

PURGEI
If on, task is deleted regardless of its condition.

PURGEU
If on, the task is deleted immediately or when the current work queue, if executing, completes. Any work queued to the task is posted as deleted.

PURGEC
If on, the task is deleted only if its work queue is empty.

PURGEW
If on, the task will be deleted when the work queue becomes empty.

STOP
Specifies the stop time option flags which define the contents of ETIME. Only one of the flags may be set. See ETIME for flag definitions.

The PTIME facilities are invoked by calling DPPPIF with the appropriate structure properly completed. Examples presented on the next pages use the default structure definitions PTIMESTR and PTIMRSTR (explained above), which are copied via %INCLUDE PTIMEDEF and %INCLUDE PTIMRDEF, respectively. Each example assumes the following PL/I statements:

```
DCL   1 PATCHSTR,
      2 MACID FIXED BIN INIT(0),       /* PATCH MACRO ID */
      2 PC FIXED BIN INIT(0),          /* RETURN CODE */
      2 PACHPARM POINTER,              /* A(PARAMETERS) */
      2 TASKNAME CHAR(8) INIT(' '),    /* TASK NAME */
      2 EPNAME CHAR(8) INIT('IEFBR14'), /* LOAD MODULE */
      2 NAME CHAR(8) INIT(' '),        /* RELATIVE TASK OF VALUE */
      2 QUEUE FIXED BIN INIT(1),       /* DEFAULT = 1 */
      2 VALUE FIXED BIN INIT(0),       /* DEFAULT = 0 */
      2 ECB POINTER,                   /* ECB ADDRESS */
      2 FREEL FIXED BIN(31,0) INIT(0), /* RESERVED */
      2 FREEA FIXED BIN(31,0) INIT(0), /* RESERVED */
      2 TCBX FIXED BIN(31,0) INIT(0) , /* TCB EXTENSION */
      2 PFLAGS,                        /* FLAG OPTIONS IF BIT IS SET ON */
          3 FO,                        /* RESERVED */
          MASTER,                      /* PARTITION=MASTER */
          SLAVE,                       /* PARTITION=SLAVE */
          F3,                          /* RESERVED */
          REPCH,                       /* ECB REPATCH */
          QPOS,                        /* QPOS=FIRST */
          DPCH,                        /* QPOS=DPATCH */
          DEL) BIT(1) INIT('0'B) ;     /* EP DELETE */

DCL   1 PATCHPRM,
      2 LENG FIXED BIN,
      2 PATID FIXED BIN,
      2 PARX (10) FIXED BIN(31,0);

DCL   1 PTIMRSTR,                      /* STRUCTURE FOR PTIME TYPE=RET */
      2 MACID FIXED BIN INIT(4),       /* PTIME SERVICE */
      2 RC FIXED BIN INIT(0),          /* RETURN CODE */
      2 TYPE FIXED BIN(31,0) INIT(0) , /* PTIME CALL TYPE */
      2 TIME FIXED BIN(31,0) INIT(0) , /* CURRENT TIME */
      2 TIMDSECT POINTER;              /* A(TIME ARRAY */

DCL   1 PTIMESTR,
      2 MACID FIXED BIN INIT(4),
      2 RC FIXED BIN INIT(0),
      2 TYPE FIXED BIN(31,0) INIT(4) , /* PTIME CALL TYPE */
      2 STIME FIXED BIN(31,0) INIT(0), /* START TIME */
      2 ITIME FIXED BIN(31,0) INIT(0), /* INTERVAL TIME */
      2 ETIME FIXED BIN(31,0) INIT(0), /* STOP TIME */
      2 PATCH POINTER,                 /* A(PATCH SUPL) */
      2 PARMS POINTER,                 /* A(PARAMETERS) */
      2 START,                         /* FLAGS DEFINE STIME CONTENTS */
          3 (F0,F1,F2,F3,F4,           /* RESERVED */
              SADJFLAG,                /* ADJUSTED TIME */
              STODFLAG,                /* TIME OF DAY */
              SRELFLAG)BIT(1) INIT('0'B), /* RELATIVE TIME */
      2 PURGE,                         /* FLAGS DEFINE PTIME PURGE OPTIONS */
          3 (F0,F1,F2,F3,              /* RESERVED */
              PURGEI,                  /* DPATCH=I */
              PURGEW,                  /* DPATCH=W */
```

```
                PURGEC,                       /* DPATCH=C */
                PURGEU) BIT(1) INIT('0'B), /* DPATCH = U */

      2 STOP,                                 /* FLAGS DEFINE ETIME CONTENTS */
          3 (F0,F1,F2,F3,                     /* RESERVED */
                ECNTFLAG,                     /* COUNT VALUE */
                EADJFLAG,                     /* ADJUSTED TIME */
                ETODFLAG,                     /* TIME OF DAY */
                ERELFLAG)BIT(1) INIT('0'B); /* RELATIVE TIME */

 DCL    1 TIMED BASED (TIMDSECT),
        2 TIMEHS FIXED BIN(31,0),
        2 TIMETOD FIXED BIN(31,0),
        2 TIMEJDAY FIXED DEC(7,0),
        2 TIMEMDAY FIXED DEC(7,0),
        2 TIMEEBC CHAR(10),
        2 TIMEBDAY FIXED BIN;
```

EXAMPLE 1: In the first example, the program uses the default structure PTIMRSTR to obtain the current time. Note, that as a result of the CALL, the time array structure TIMED is usable since its base variable (a POINTER variable in PTIMRSTR) has been set. The current time is used to set the start time in PTIMESTR for PATCHes by PTIME, at current time plus 1 hour. The interval is set to 1 hour, and the last PATCH is to occur 3 hours later. The PATCH parameters are set to create the task TIMETEST with a work queue length of 5, and a dispatching priority of 15 less than the PTIME task. The PATCH will execute program TTEST and delete it when the processing of each work request completes. The parameters passed are day of the year and time of the PTIME request with a PATCH ID of 10.

```
 CALL DPPPIF(PTIMRSTR.MACID);            /* CURRENT TIME */
 PATCH = ADDR (PATCHSTR.TASKNAME) ;      /* BUILD THE PTIME */
 PARMS = ADDR (PATCHPRM.LENG);           /* PARAMETERS */
 STIME = TIME+360000;
 STODFLAG = '1'B;
 ITIME = 360000;
 ETIME = STIME+1080000;
 ETODFLAG = '1'B;
 TASKNAME = 'TIMETEST';                  /* BUILD THE PATCH */
 QUEUE = 5;                              /* PARAMETERS */
 VALUE = 15;
 EPNAME = 'TTEST';
 DEL = '1'B;
 LENG = 12;                              /* BUILD THE PROGRAM */
 PATID = 10;                             /* PARAMETERS */
 PARX(1) = TIMEBDAY;
 PARX(2) = TIME;
 CALL DPPPIF(PTIMESTR.MACID);            /* ISSUE THE PTIME */
```

EXAMPLE 2: For the second example, the PTQE built by Example 1 will be modified (TYPE = 8) to start the PATCHes 15 seconds after this PTIME is issued, the interval to once a minute, and the stop time to never end. The program will not be deleted when a work request is finished processing and the work request will be queued first. The PATCH ID will be changed to 5. Note, that all parameters must be re-specified, as a modify acts as a replace. All structures are initially default.

```
 TYPE = 8;                              /* MODIFY PTQE */
 PATCH = ADDR(PATCHSTR.TASKNAME);
 PARMS = ADDR(PATCHPRM.LENG);
 STIME = 1500;
 SRELFLAG = '1'B;
 ITIME = 6000;
 TASKNAME = 'TIMETEST';                 /* BUILD PATCH PARAMETERS */
 QUEUE = 5;
 VALUE = 15;
 EPNAME = 'TTEST';
 QPOS = 1;
 LENG = 12;                             /* BUILD PROGRAM PARAMETERS */
 PATID = 5,
 PARX(1) = TIMEBDAY;
 PARX(2) = TIME;
 CALL DPPPIF (PTIMESTR.MACID);          /* ISSUE PTIME */
```

EXAMPLE 3: Example 3 shows the use of the adjusted time facility of PTIME. The first PATCH is to occur at 5 a.m. or within 30 minutes of when the PTIME was issued and at 30-minute intervals for 6 times. The task is to be deleted immediately when the PTQE is destroyed.

```
 PURGEU = '1'B;                         /* BUILD PTIME PARAMETERS */
 STIME = 1800000;
 SADJFLAG = '1'B;
 ITIME = 180000;
 ETIME = 6;
 ECNTFLAG = '1'B;
    PATCH PARAMETERS
           .
           .
           .

    PROBLEM PARAMETERS
           .
           .
           .
 CALL DPPPIF(PTIMESTR.MACID);           /* ISSUE PTIME */
```

EXAMPLE 4: Example 4 is the example for deleting a PTQE. Since the function of this PTIME service request is to locate the PTQE which is to be destroyed, only the parameters required to identify the PTQE need be given. In this case, the task is to be DPATCHed as well.

```
 PURGEU = '1'B;
 TYPE = 12;
 PATCH = ADDR(TASKNAME);
 PARMS = ADDR(LENG);
 TASKNAME = 'TIMETEST';
 EPNAME = 'TTEST';
 PATID = 10;
 CALL DPPPIF(PTIMESTR.MACID);
```

This example would remove the PTQE created by Example 1.

PL/I-DPATCH Interface

The Special Real Time Operating System DPATCH facility provides the programmer the method for destroying tasks which were created by the PATCH service.

A PL/I interface exists to provide a DPATCH service. The default structure, DPACHSTR, shown below, may be copied into the PL/I program by a %INCLUDE DPACHDEF.

```
DCL   1 DPACHSTR,                       /* DPATCH STRUCTURE */
      2 MACID FIXED BIN INIT(8)         /* DPATCH ID */
      2 RC FIXED BIN INIT(0),           /* RETURN CODE */
      2 TYPE FIXED BIN INIT(0),         /* DEFAULT PURGE = U */
      2 TASK CHAR(8) INIT(' ');         /* TASK NAME */
```

DPACHSTR
Specifies the name of the default structure used to destroy tasks created by a PATCH.

MACID
Specifies a halfword binary value set to 8 to identify a DPATCH service request.

RC
Specifies a halfword binary value containing the return code from the service request. The return codes are defined in the macro description.

TYPE
Specifies halfword binary value specifying the DPATCH service requests. If 0 is specified, the task is deleted immediately or at the completion of the currently executing work request. Any work queued to the task is posted as deleted. If 4 is specified, the task is deleted only if its work queue is empty. If 8 is specified, the task is deleted when the work queue becomes empty. This does not prevent new work from being queued. If 12 is specified, the task is deleted even if it is active.

TASK
Specifies the name of the task being deleted. If left blank, the current task is deleted.

The example assumes the above default structure. The first DPATCH request sets up the current task to be deleted when its work queue becomes empty. The second DPATCH requests that the task be deleted only if it is not doing any work. The last DPATCH requests that the task be destroyed regardless of its condition.

```
TYPE = 8;
CALL DPPPIF(DPACHSTR.MACID);
TYPE = 4;
TASK = 'TESTDPCH';
CALL DPPPIF(DPACHSTR.MACID);
TYPE = 12;
TASK = 'DPCHTEST';
CALL DPPPIF(DPACHSTR.MACID);
```

PL/I MESSAGE Interface

The MESSAGE service is used to cause a predefined message to be printed or displayed. The message must have been defined through the offline utility system using the DEFMSG macro.

The PL/I structure, MESAGSTR, (defined below) contains the parameters for the MESSAGE service and may be copied into the program via %INCLUDE MESAGDEF;

```
DCL    1 MESAGSTR,
       2 MACID FIXED BIN INIT(40),
       2 RC FIXED BIN INIT(0),
       2 MSGNUM FIXED BIN INIT(0),           /* MESSAGE NUMBER */
       2 ACT CHAR(1) INIT(' '),              /* ACTION CODE */
       2 WAIT BIT(1) INIT('0'B),             /* WAIT = NO */
       2 RESERVED FIXED BIN(31,0) INIT(0)    /* RESERVED */
       2 AREA POINTER,                       /* A(RETURN OF MESSAGE) */
       2 ROUTE (8) FIXED BIN INIT(0),        /* ROUTING CODES */
       2 VAR (10) POINTER;                   /* A(VARIABLES) ARRAY */
```

MESAGSTR
Is the name of the default structure used for the PL/I message interface.

MACID
Is a halfword binary value of 40 to indicate the service requested to the interface routine.

RC
Is a halfword binary value containing the return code from the service routine. See MESSAGE macro for possible values.

MSGNUM
Is a halfword binary value from 1 to 999 identifying the message requested.

ACT
Is a 1-byte character to be appended to the message number. I denotes information; A denotes action is required; and D denotes that a decision is required.

WAIT
Is a flag bit indicating the program's decision to WAIT for the message to be sent. Default is off, which is no wait.

RESERVED
Is a fullword binary field reserved for the interface routine.

AREA
Is a pointer variable containing the address of an area where the service routine will place the formatted message for use by the program.

ROUTE
Specifies a table of 8 halfword binary numbers representing the devices on which the message will appear or will be printed. All unused entries must be zero.

VAR
Specifies a table of 10 pointer variables addressing the variable data to be converted and inserted into the message. All unused entries must be zero. Only consecutive non-zero entries will be used.

The example below requests the MESSAGE service to output to routing code (1) message number 37 with a variable text field of "JOB IS FINISHED, PLEASE CANCEL". The message number will have an action code of "A" appended to notify the operator to act. The program will wait for the message to be transmitted. The example presumes the above MESAGSTR structure.

```
%INCLUDE MESAGDEF;
      .
      .
      .
DCL A CHAR(50)
      INIT('JOB IS FINISHED. PLEASE CANCEL');

DCL X CHAR(128);
MSGNUM = 37;
ACT = 'A';
WAIT = '1'B;
AREA = ADDR(X);
ROUTE(1) = 1;
VAR(1) = ADDR(A);
VAR(2) = NULL;
CALL DPPPIF(MESAGSTR.MACID);
```

PL/I-RECORD Interface

The RECORD facility provides a method for writing data to a sequential data set. The data can be retrieved at a later time for offline processing.

The default PL/I structure RECRDSTR, defined below, can be copied into the program via a %INCLUDE RECRDDEF;

```
DCL    1 RECRDSTR,
       2 MACID FIXED BIN INIT(56),            /* RECORD ID */
       2 RC FIXED BIN INIT(0),                /* RETURN CODE */
       2 COUNT FIXED BIN(31,0) INIT(0),       /* DATA LENGTH */
       2 DATX POINTER,                        /* DATA ADDRESS */
       2 ID FIXED BIN INIT(0);                /* DATA ID NO. */
```

RECRDSTR
Is the name of the default structure used to invoke the RECORD service.

MACID
Is a halfword binary number used to identify the service being requested. Default is 56 for RECORD.

RC
Is a halfword binary value containing the completion code from the RECORD service routine. See RECORD macro writeup for valid return codes.

COUNT
Is a fullword binary number which is the number of bytes to be recorded. A maximum value of 65535 bytes may be specified.

DATX
Is the address of the data to be recorded.

ID
Is a halfword binary number from 1 to 4095 which identifies the data being recorded.

The following example presumes the RECRDSTR structure above:

```
DCL A (16) FIXED BIN INIT(5);
COUNT = 32;
ID = 10;
DATX = ADDR(A);
CALL DPPPIF(RECRDSTR.MACID);
```

PL/I PATCH-WAIT Interface

This interface provides the PL/I programmer the facility to wait for the completion of a work queue element generated by a PATCH. The following default structure WAITSTR may be copied into a PL/I program by a %INCLUDE WAITDEF.

```
DCL    1 WAITSTR,                          /* PATCH-WAIT STRUCTURE */
       2 MACID FIXED BIN INIT(60),         /* WAIT MACRO ID */
       2 RC FIXED BIN INIT(0),             /* ECBPOST CODE */
       2 EVENT FIXED BIN(31,0) INIT(0);    /* ECB */
```

WAITSTR
Is the name of the default structure provided for waiting on PATCH request completion.

MACID
Is a halfword binary number of 60 identifying the service requested to the interface routine.

RC
Is a halfword binary number containing the completion flag byte from POST. See PATCH macro for possible values.

EVENT
Is a fullword binary field containing the completion code from the finished work queue processing or the address of a REPATCH control block. The value in this field is governed by the contents of RC.

Note: For this structure, RC will never be zero when the interface routine returns to the PL/I program.

The following example uses the default structures for PATCHSTR and WAITSTR as shown. Note, that the user need not zero the variable EVENT as the interface routine will automatically zero the first byte when moving it to the RC field.

```
DCL  1 PATCHPRM,
     2 LENG FIXED BIN,
     2 PATID FIXED BIN,
     2 PARX (10) FIXED BIN(31,0);

DCL  1 PATCHSTR,
     2 MACID FIXED BIN INIT(0),
     2 RC FIXED BIN INIT(0),
     2 PACHPARM POINTER,
     2 TASKNAME CHAR(8) INIT(' '),
     2 EPNAME CHAR(8) INIT('IEFBR14'),
     2 NAME CHAR(8) INIT(' '),
     2 QUEUE FIXED BIN INIT(1),
     2 VALUE FIXED BIN INIT(0),
     2 ECB POINTER,
     2 FREEL FIXED BIN(31,0) INIT(0),
     2 FREEA FIXED BIN(31,0) INIT(0),
     2 PCBX FIXED BIN(31,0)  INIT(0),
     2 PFLAGS,
        3 (F0,
        MASTER,
        SLAVE,
        F3,
        REPCH,
        QPOS,
        DPCH,
        DEL) BIT(1)  INIT('0'B);

DCL  1 WAITSTR,
     2 MACID FIXED BIN INIT(60),
     2 RC FIXED BIN INIT(0),
     2 EVENT FIXED BIN(31,0)     INIT(0);
LENG = 4;
PATID = 2;
PACHPARM = ADDR(PATCHPRM.LENG);
TASKNAME = 'TESTWAIT';
EPNAME = 'WAITTEST';
ECB = ADDR(EVENT);
CALL DPPPIF(PATCHSTR.MACID);
     2 EVENT FIXED BIN(31,0) INIT(0);
```

PL/I REPATCH Interface

This PL/I interface provides the programmer the facilities of the Special Real Time Operating System REPATCH service. The default structure, REPCHSTR (defined below), may be copied into the PL/I program via a %INCLUDE REPCHDEF;.

```
DCL   1 REPCHSTR,                         /* REPATCH STRUCTURE */
      2 MACID FIXED BIN INIT(12),         /* REPATCH MACRO ID */
      2 RC    FIXED BIN INIT(0),          /* RETURN CODE */
      2 TYPE  FIXED BIN(31,0) INIT(0),    /* SERVICE TYPE */
      2 REPCB FIXED BIN(31,0),            /* A(REPATCH CNTL BLK) */
      2 TASK  CHAR(8),                    /* TASKNAME */
      2 EP    CHAR(8)                     /* LOAD MODULE */
      2 RELTASK CHAR(8),                  /* REL TASK FOR VALUE */
      2 QUE   FIXED BIN,                  /* QUEUE LENGTH */
      2 VAL   FIXED BIN,                  /* PRIORITY CHG */
      2 ECB   POINTER,                    /* ECB ADDRESS */
      2 RES (2) POINTER,                  /* RESERVED */
      2 TCBX POINTER,                     /* TCBX ADDRESS */
      2 PFLAGS,                           /* FLAG OPTIONS IF BIT IS SET ON */
        3 (FO,                            /* RESERVED */
        MAST,                             /* PATCH PARTITION = MASTER */
        SLAV,                             /* PATCH PARTITION = SLAVE */
        F3,                               /* RESERVED */
        RPECB,                            /* ECB REPATCH */
        QPOS1,                            /* QPOS=FIRST */
        DPACH,                            /* QPOS=DPATCH */
        DELET) BIT(1),                    /* EP DELETE */
      2 RES1 (3) POINTER;                 /* RESERVED SUPERVISOR POINTERS */
```

REPCHSTR
Name of the default structure provided for the Special Real Time Operating System REPATCH service requests.

MACID
A halfword binary value of 12 identifying the service required to the interface routine.

RC
A halfword field containing a binary number return code from the REPATCH/PATCH service routine. See REPATCH macro write-up for REPATCH and related PATCH return codes.

TYPE
A fullword binary value indicating the interface routine service required.

0 -- The REPATCH control block is to be copied to the REPCHSTR to permit alteration of PATCH parameters prior to REPATCH.

4 -- Issue REPATCH TYPE=EXEC.

8 -- Issue REPATCH TYPE=PURGE.

REPCB
A fullword binary field to contain the REPATCH control block address placed in the WAITSTR.EVENT when WAITSTR.RC equaled 68. The value in EVENT must be moved to REPCB before any interface call except the first interface call TYPE=4 or 8 following a TYPE=0 interface call.

TASK
Specifies an 8-character name which is the name of the task being referenced by this PATCH.

EP
Specifies the 8-character valid program name of the program to be scheduled under the task specified in TASK.

RELTASK and VAL
Specifies an 8-character task name and a halfword value which will determine the priority of the new task. VAL will be subtracted from

the dispatching priority of the specified task. VAL may range from 0 to 255 with zero default. See PRTY option of PATCH macro for further detail.

QUE
A halfword value specifying the number of work queue entries to be provided for a new independent task.

ECB
Specifies the address of the ECB within a WAITSTR which is to be used in a CALL DPPPIF. This ECB is posted when processing for this PATCH completes. The ECB which contained the REPATCH control block address may be reused and will be if this parameter is left unchanged.

TCBX
Specifies the address of the TCB extension control block for an existing independent task.

PFLAGS
The PATCH option flags as described below:

MAST
This PATCH is intended for the MASTER partition.

SLAV
This PATCH is intended for the SLAVE partition.

RPECB
Specifies that if this work request is pushed off the queue, the ECB is to be posted with a REPATCH control block address.

QPOS1 and DPACH
Specifies where in the task work queue this work request is to go if the task is busy. If QPOS1 is on, the request is to be placed first on the queue. If DPACH is on, the processing for this PATCH will not be executed until a DEPATCH is issued for this task. Both flags off means this request is queued last.

DELET
Specifies that a DELETE is issued for the EP name after processing completes for this PATCH.

RES and RES1
The pointers must remain unchanged.

The Special Real Time Operating System REPATCH service may be invoked by including the REPCHDEF in the PL/I program, moving the REPATCH control block address from the event control block to REPCB and then executing one of the following:

a. If the REPATCH is to be done without change, set TYPE to 4 or 8 and CALL DPPPIF.

b. If the REPATCH is to be changed prior to execution, set TYPE to 0, CALL DPPPIF, make changes desired, set TYPE to 4 and CALL DPPPIF again.

Users of this facility should be aware that only the "supervisor" portion of the PATCH parameters can be altered. The problem parameters cannot be changed. All REPATCH control blocks must be returned to the system through a TYPE=4 or 8 service request.

The following examples will show the various methods of using REPCHSTR.

The examples for using the REPCHSTR use the following set of structures:

```
  DCL  1 REPCHSTR,                      /* REPATCH STRUCTURE */
       2 MACID FIXED BIN INIT(12) ,     /* REPATCH MACRO ID */
       2 RC    FIXED BIN INIT(0),       /* RETURN CODE */
       2 TYPE  FIXED BIN(31,0) INIT(0), /* SERVICE TYPE */
       2 REPCB FIXED BIN(31,0),         /* A(REPATCH CNTL BLK) */
       2 TASK  CHAR(8),                 /* TASKNAME */
       2 EP    CHAR(8),                 /* LOAD MODULE */
       2 RELTASK CHAR(8),               /* REL TASK FOR VALUE */
       2 QUE   FIXED BIN,               /* QUEUE LENGTH */
       2 VAL   FIXED BIN,               /* PRIORITY CHG */
       2 ECB   POINTER,                 /* ECB ADDRESS */
       2 RES (2) POINTER,               /* RESERVED */
       2 TCBX POINTER,                  /* TCBX ADDRESS */
       2 PFLAGS,                        /* FLAG OPTIONS IF BIT IS SET ON */
            3 (F0,                      /* RESERVED */
            MAST,                       /* PATCH PARTITION = MASTER */
            SLAV,                       /* PATCH PARTITION = SLAVE */
            F3,                         /* RESERVED */
            RPECB,                      /* ECB REPATCH */
            QPOS1,                      /* QPOS=FIRST */
            DPACH,                      /* QPOS=DPATCH */
            DELET) BIT(1),              /* EP DELETE */
       2 RES1 (3) POINTER;              /* RESERVED SUPERVISOR POINTERS */
  DCL  1 WAITSTR,                       /* PATCH-WAIT STRUCTURE */
       2 MACID FIXED BIN INIT(60) ,     /* WAIT MACRO ID */
       2 RC FIXED BIN INIT(0),          /* ECB POST CODE */
       2 EVENT FIXED BIN(31,0) INIT(0); /* ECB */
```

EXAMPLE 1: Example 1 shows the correct method for purging a REPATCH control block, should a work request fail to be executed. The example begins with the PATCH-WAIT which is notified about the work request not getting done.

```
                   .
                   .
                   .
    CALL DPPPIF (WAITSTR.MACID);
    IF WAITSTR.RC = 68 THEN DO;
       REPCHSTR.REPCB = WAITSTR.EVENT;
       REPCHSTR.TYPE = 8;
       CALL DPPPIF (REPCHSTR.MACID);
    END;
```

Example 1

EXAMPLE 2: Example 2 demonstrates the method for altering a REPATCH control block. As with Example 1, this example begins with a WAIT on a PATCH.

```
                    .
                    .
                    .
X:  CALL DPPPIF (WAITSTR.MACID);
    IF WAITSTR.RC = 68 THEN DO;
       REPCHSTR.REPCB = WAITSTR.EVENT;
       REPCHSTR.TYPE = 0;
       CALL DPPPIF (REPCHSTR.MACID);
       REPCHSTR.PFLAGS.QPOS1 = '1'B;
       WAITSTR.EVENT = 0;
       REPCHSTR.TYPE = 4;
       CALL DPPPIF (REPCHSTR.MACID);
       IF REPCHSTR.RC <8 THEN GOTO X;
    END;
```

Example 2

The above example replaces the work request on the work queue for the same task as previously requested, except that it will be placed first on the queue.

PL/I GETARRAY/PUTARRAY Interface

This PL/I interface provides the programmer the facilities of the Special Real Time Operating System GETARRAY and PUTARRAY services. The default structure, ARRAYSTR (defined below), may be copied into the PL/I program via a %INCLUDE ARRAYDEF;.

```
DCL  1 ARRAYSTR,                  /* GET/PUT ARRAY STRUCTURE */
     2 MACID FIXED BIN INIT(16),  /* ARRAY MACRO ID */
     2 RC FIXED BIN INIT(0),      /* RETURN CODE */
     2 NAME POINTER,              /* A(NAMELIST/NUMBERLIST/ADDRLIST) */
     2 AREA POINTER,              /* A(FINDLIST/DATAAREALIST) */
     2 NAMEINCR FIXED BIN INIT(0), /* LIST INCREMENT */
     2 AREAINCR FIXED BIN INIT(0), /* LIST INCREMENT */
     2 TYPE FIXED BIN INIT(0);    /* TYPE OF ARRAY SERVICE */
```

ARRAYSTR
Name of the default structure provided for the Special Real Time Operating System array service requests.

MACID
A halfword binary value of 16 identifying the service required to the interface routine.

RC
A halfword field containing a binary number return code from the array service routine. See GETARRAY and PUTARRAY macro write-ups for possible values.

NAME
The address of one of the following based on the specifications implied by the value of TYPE.

a. If TYPE specifies 'NAMELIST', then NAME points to a list of 8-character array names followed by an X'FF' after the last name where the next name would start. NAMEINCR contains the value to be added to the list address to locate the next array name.

NAME LIST

0	NAME1
8	NAME2
16	FF

b. If TYPE specifies 'NUMBERLIST', then NAME points to a list of halfword binary array numbers followed by an X'FF' after the last array number where the next number would start. NAMEINCR contains the value to be added to the list address to locate the next array number in the list.

NUMBER LIST

0	1ST NUMBER	
2	2ND NUMBER	
4	FF	

c. If TYPE specifies 'ADDRESSLIST', then NAME points to a list of array addresses as returned from a previous GETARRAY execution. The list must be terminated by a fullword binary value of -1 after the last array address where the next address would be located. NAMEINCR contains the value to be added to the list address to locate the next array address.

ADDRESS LIST

0	A(1ST ARRAY)
4	A(2ND ARRAY)
8	FFFFFFFF

AREA
The address of one of the following based on the specifications implied by the value of TYPE.

a. If TYPE specifies 'DATALIST', then AREA points to a list of addresses into or from which the data of the specified arrays (see NAME above) is to be moved. AREAINCR contains the value to be added to the list address to locate the next data area address in the list.

DATA AREA ADDRESS LIST

0	A(1ST DATA AREA)
4	A(2ND DATA AREA)
8	A(3RD DATA AREA)

b. If TYPE specifies 'FINDLIST', then AREA points to a list of 10-byte fields to be filled with a flag byte (see GETARRAY macro write-up), a 3-byte array address, a halfword block count, a halfword array size or block size and a halfword item count. The list must contain one entry more than the number of addresses expected to allow for an end of list X'FF'. AREAINCR contains the value to be added to the list address to locate the next 10-byte field. The minimum value for AREAINCR under this option is 8; in which case, the item count halfword will not be in the list.

FIND LIST

0	FLG	ARRAY ADDR	NO.BLKS	SIZE	NO.ITEMS
10	FLG	ARRAY ADDR	NO.BLKS	SIZE	NO.ITEMS
20	FF				

c. If TYPE specifies 'SPECLIST', then AREA points to a list of 16-byte fields to be filled with an 8-byte item name, a 1-byte item length, a 1-byte data type, a halfword array displacement to the start of the item, a halfword array ID, and a halfword number identifying the number of identical and sequential items defined by this entry. AREAINCR contains the value to be added to the list address to locate the next 16-byte field.

ARRAY SPECIFICATIONS LIST

0	ITEM NAME	LNG	TYPE	DISP.	AID	REPT
16	ITEM NAME	LNG	TYPE	DISP	AID	REPT
32	ITEM NAME	LNG	TYPE	DISP	AID	REPT

NAMEINCR
A halfword value added to NAME to locate the next entry in the list. A value must be specified.

AREAINCR
A halfword value added to AREA to locate the next entry in the list. A value must be specified.

TYPE
A halfword binary value specifying the array service options selected. The values (given in the tables below) identify the contents of NAME and AREA, either a GETARRAY or PUTARRAY, the array service (i.e., DATALIST, ADDRLIST or SPECLIST), and the desired protection for GETARRAYs (PROTECT or RISK).

DATALIST
Specifies that the content of the array(s) is to be returned (GETARRAY) or updated (PUTARRAY).

ADDRLIST
Specifies that a 'FINDLIST' entry is to be completed for each array name or number in the list. Option is valid for virtual storage resident arrays only.

SPECLIST
Specifies that a 'SPECLIST' entry is to be completed for each item of each array name or number in the list.

PROTECT
Specifies that the array service will lock during processing to prevent changes from altering results.

RISK
Specifies that the array service will be processed regardless of the possibility of parallel processing changing the array content.

NAME	AREA	SERVICE REQUESTED	PROTECTION REQUESTED	TYPE VALUE
A(NAME LIST)	A(DATA LIST)	DATA LIST	PROTECT	16
A(NAME LIST)	A(DATA LIST)	DATA LIST	RISK	17
A(NAME LIST)	A(SPEC LIST)	SPECLIST	PROTECT	20
A(NAMELIST)	A(SPEC LIST)	SPECLIST	RISK	21
A(NAME LIST)	A(FIND LIST)	ADDR LIST	PROTECT	34
A(NAME LIST)	A(FIND LIST)	ADDR LIST	RISK	35
A(ADDR LIST)	A(DATA LIST)	DATA LIST	PROTECT	48
A(ADDR LIST)	A(DATA LIST)	DATA LIST	RISK	49
A(NUMBER LIST)	A(DATA LIST)	DATA LIST	PROTECT	80
A(NUMBER LIST)	A(DATA LIST)	DATA LIST	RISK	81
A(NUMBER LIST)	A(SPEC LIST)	SPECLIST	PROTECT	84
A(NUMBER LIST)	A(SPEC LIST)	SPECLIST	RISK	85
A(NUMBER LIST)	A(FIND LIST)	ADDR LIST	PROTECT	98
A(NUMBER LIST)	A(FIND LIST)	ADDR LIST	RISK	99

Figure 2-15. GETTARRAY Services

A(NAME LIST)	A(DATA LIST)	DATA LIST	N/A	128
A(ADDR LIST)	A(DATA LIST)	DATA LIST	N/A	144
A(NUMBER LIST)	A(DATA LIST)	DATA LIST	N/A	176

Figure 2-16. PUTARRAY Services

The GETARRAY/PUTARRAY services are invoked in PL/I by CALLing DPPPIF with the properly completed array name/number/address list data address list and structure (ARRAYSTR or a similar structure).

The examples for using GETARRAY or PUTARRAY services in PL/I use the following list of structures and variables:

```
    DCL   1 ARRAYSTR,
          2 MACID FIXED BIN INIT(16),
          2 RC FIXED BIN INIT(0),
          2 NAME POINTER,
          2 AREA POINTER,
          2 NAMEINCR FIXED BIN INIT(0),
          2 AREAINCR FIXED BIN INIT(0),
          2 TYPE FIXED BIN INIT(0);
    DCL   1 ARRAY,
          2 NAME (2) CHAR(8), NO (2) FIXED BIN,
          2 FIND (2),
          3 ADDRESS POINTER,
          3 BLKCNT FIXED BIN,
          3 BLKSIZ FIXED BIN,
          3 ITEMCNT FIXED BIN,
          3 RES FIXED BIN,
          2 CORE (2) POINTER;
    DCL   1 ARRAYITM (255),
          2 NAME CHAR(8),
          2 LNG BIT(8),
          2 TYP BIT(8),
          2 DISP FIXED BIN;
    DCL ITEM (255) CHAR(16);
    DCL Q POINTER BASED(P);
```

Note, that the structure ARRAY has the field NAME for use when calling arrays by name, or NO for use when calling arrays by number.

Both of the following examples make use of the fact that once a structure has been altered it remains unchanged; i.e., the array name in the first example needed to be specified only once.

The first example will locate array 'B' through the FINDLIST option, read in the item specifications through the SPEC option and then read in the array. The array is then changed and the new array transmitted.

```
      ARRAY.NAME(1) = 'B  ';                       /* BUILD PARAMETER */
      P = ADDR(ARRAY.NAME(2));                     /* LIST TO LOCATE */
      Q = NULL;                                    /* NAMED ARRAY */
      ARRAYSTR.NAME = ADDR(ARRAY.NAME(1));
      ARRAYSTR.NAMEINCR = 8;
      ARRAYSTR.AREA = ADDR(ARRAY.FIND(1).ADDRESS);
      ARRAYSTR.AREAINCR = 12;
      ARRAYSTR.TYPE = 35;
      CALL DPPPIF(ARRAYSTR.MACID);
  /* THE FIND LIST HAS BEEN BUILT */
      ARRAY.CORE(1) = ADDR(ARRAYITM(1).NAME);      /* BUILD PARAMETER */
      ARRAYSTR.AREAINCR = 4;                       /* LIST TO */
      ARRAYSTR.AREA = ADDR(ARRAY.CORE(1));         /* OBTAIN */
      ARRAYSTR.TYPE = 21;                          /* LIST OF ITEM NAMES */
      CALL DPPPIF(ARRAYSTR.MACID);
  /* THE ITEM SPECIFICATIONS HAVE BEEN OBTAINED */
      ARRAY.CORE(1) = ADDR(ITEM(1));               /* READ THE */
      ARRAYSTR.TYPE = 16;                          /* ENTIRE ARRAY */
      CALL DPPPIF(ARRAYSTR.MACID);
  /* THE ARRAY HAS BEEN READ */
      ITEM(1) = 'THIS BLOCK ZAPED';
  /* THE ARRAY IS ALTERED */
      ARRAYSTR.TYPE = 128;                         /* WRITE THE ENTIRE ARRAY */
      CALL DPPPIF(ARRAYSTR.MACID);
  /* THE ARRAY IS UPDATED */
```

Example 1

Note that in the above example all services to the array are by name.

The second example is identical to the first, except that the array is numbered.

```
    ARRAY.NO(1) = 1;
    ARRAY.NO(2) = -1;
    ARRAYSTR.NAME = ADDR(ARRAY.NO(1));
    ARRAYSTR.NAMEINCR = 2;
    ARRAYSTR.AREA = ADDR(ARRAY.FIND(1).ADDRESS);
    ARRAYSTR.AREAINCR = 12;
    ARRAYSTR.TYPE = 99;
    CALL DPPPIF(ARRAYSTR.MACID);
/* THE FINDLIST HAS BEEN BUILT */
    ARRAY.CORE(1) = ADDR(ARRAYITM(1).NAME);
    ARRAYSTR.AREA = ADDR(ARRAY.CORE(1));
    ARRAYSTR.AREAINCR = 4;
    ARRAYSTR.TYPE = 85;
    CALL DPPPIF(ARRAYSTR.MACID);
/* THE ITEM SPECIFICATIONS HAVE BEEN OBTAINED */
    ARRAY.CORE(1) = ADDR(ITEM(1));
    ARRAYSTR.NAME = ADDR(ARRAY.FIND(1).ADDRESS);
    ARRAY.FIND(2).ADDRESS = NULL;
    /ARRAYSTR.NAMEINCR = 12;
    ARRAYSTR.TYPE = 48;
    CALL DPPPIF(ARRAYSTR.MACID);
/* THE ARRAY HAS BEEN READ */
    ITEM(1) = 'THIS BLOCK ZAPED';
/* THE ARRAY IS ALTERED */
 ARRAYSTR.TYPE = 144;
    CALL DPPPIF(ARRAYSTR.MACID);
/* THE ARRAY HAS BEEN UPDATED */
```

Example 2

Note, that the array was read and written using the ADDR option.

PL/I GETITEM/PUTITEM Interface

This PL/I interface provides the programmer the facilities of the Special Real Time Operating System GETITEM and PUTITEM services. The default structure, ITEMSTR (defined below), may be copied into the PL/I program via a %INCLUDE ITEMDEF;.

```
 DCL   1 ITEMSTR,                      /* GET/PUT ITEM STRUCTURE */
       2 MACID FIXED BIN INIT(20),     /* ITEM MACRO ID */
       2 RC FIXED BIN INIT(0),         /* RETURN CODE */
       2 NAME POINTER,                 /* A(NAMELIST/ADDRLIST) */
       2 AREA POINTER,                 /* A(DATA AREA) */
       2 NAMEINCR FIXED BIN INIT(0),   /* LIST INCREMENT */
       2 AREAINCR FIXED BIN INIT(0),   /* LIST INCREMENT */
       2 TYPE FIXED BIN INIT(0);       /* TYPE OF ITEM SERVICE */
```

ITEMSTR
Name of the default structure provided for the Special Real Time Operating System array-item service requests.

MACID
A halfword binary value of 20 identifying the service required to the interface routine.

RC
A halfword field containing a binary number return code from the item service routine. See GETITEM and PUTITEM macro write-ups for possible values.

NAME
The address of one of the following based on the specifications implied by the value of TYPE.

a. If TYPE specifies 'NAMELIST', then NAME points to a list of 8-character item names followed by a X'FF' after the last name where the next name would start. NAMEINCR contains the value to be added to the list address to locate the next item name.

NAME LIST

0	NAME 1	
8	NAME 2	
16	F F	

b. If TYPE specifies 'ADDRESS LIST', then NAME points to a list of item addresses as returned from a previous execution. The list must be terminated by a fullword of -1 where the next address would be in the list. NAMEINCR contains the value to be added to the list address to locate the next address in the list.

ADDRESS LIST

0	A(ITEM a)
4	A(ITEM b)
8	F F F F F F F F

AREA
the address of one of the following based on the specifications implied by the value of TYPE.

a. If TYPE specifies 'DATALIST', then AREA points to a data area into or from which item data is moved. AREAINCR contains the value to be added to the area address to locate the next area for the next item. If AREAINCR is zero, then the item length is used to determine the location for the next item data area.

b. If TYPE specifies 'ADDRLIST', then AREA points to a list of 4-byte entries into which the item length and address are stored for each item specified in the 'NAMELIST'. The list must be one entry longer than the number of addresses being obtained to allow the service routine to store an end of list X'FF'. AREAINCR contains the value to be added to the area address to locate the next entry.

ADDRESS LIST

0	ITEM LENGTH	ITEM ADDRESS
4	ITEM LENGTH	ITEM ADDRESS
8	F F	F F F F F F

c. If TYPE specifies 'SPECLIST', then AREA points to a list of 4-byte entries containing the item length, flags identifying data type and a displacement into the array to the first byte of the item. AREAINCR contains the value to be added to the area address to locate the next entry.

ITEM SPECIFICATIONS LIST

0	ITEM LENGTH	TYPE FLAGS	ARRAY DISPLACEMENT
4	ITEM LENGTH	TYPE FLAGS	ARRAY DISPLACEMENT
8	ITEM LENGTH	TYPE FLAGS	ARRAY DISPLACEMENT

NAMEINCR
A halfword binary value added to the list address in NAME to locate the next entry. A value must be specified.

AREAINCR
A halfword binary value added to the list address in AREA to locate the next entry. A value must be specified unless TYPE specifies 'DATALIST' in which case zero may be used.

TYPE
A halfword binary number specifying the item service options selected. The values (given in the tables below) identify the kind of service (i.e., DATA, ADDR or SPEC), and whether it is a GETITEM with or without protection (PROTECT or RISK) or a PUTITEM.

DATALIST
Specifies that the content of the array-item is to be moved from the array to AREA or updated by the contents of AREA.

ADDRLIST
Specifies that the item 'ADDRESSLIST' is to be built in AREA for each named item.

SPECLIST
Specifies that the item 'SPECIFICATION LIST' is to be built in AREA for each named item.

PROTECT
Specifies that the GETITEM service will ensure data integrity during processing.

RISK
Specifies that the GETITEM service will process the request regardless of the possibility of parallel processing updating the content of the named item(s).

Note: DATALIST and DDDRLIST are invalid service requests for direct access resident arrays.

NAME	AREA	SERVICE REQUESTED	PROTECTION REQUESTED	TYPE VALUE
A(NAME LIST)	A(DATA LIST)	DATA LIST	PROTECT	136
A(NAME LIST)	A(DATA LIST)	DATA LIST	RISK	137
A(NAME LIST)	A(ADDR LIST)	ADDR LIST	PROTECT	138
A(NAME LIST)	A(ADDR LIST)	ADDR LIST	RISK	139
A(NAME LIST)	A(SPEC LIST)	SPEC LIST	PROTECT	140
A(NAME LIST)	A(SPEC LIST)	SPEC LIST	RISK	141
A(ADDR LIST)	A(DATA LIST)	DATA LIST	PROTECT	152
A(ADDR LIST)	A(DATA LIST)	DATA LIST	RISK	153

Figure 2-17. GETITEM Services

A(NAME LIST)	A(DATA LIST)	DATA LIST	N/A	184
A(ADDR LIST)	A(DATA LIST)	DATA LIST	N/A	200

Figure 2-18. PUTITEM Services

The GETITEM/PUTITEM services are invoked in PL/I by CALLing DPPPIF with the properly completed item name/address list, data address list and structure (ITEMSTR or a similar structure).

The example for using GETITEM or PUTITEM services in PL/I uses the following list of structures and variables:

```
    DCL     1 ITEMSTR,
            2 MACID FIXED BIN INIT(20),
            2 RC FIXED BIN INIT(0),
            2 NAME POINTER,
            2 AREA POINTER,
            2 NAMEINCR FIXED BIN INIT(0),
            2 AREAINCR FIXED BIN INIT(0),
            2 TYPE FIXED BIN INIT(0);
    DCL     1 ITEMLIST(6),
            2 NAME CHAR(8),
            2 ADR POINTER,
            2 LNG BIT(8),
            2 FLGS BIT(8),
            2 DISP FIXED BIN;
    DCL ITEM(5) CHAR(16);
    DCL Q POINTER BASED(P);
    DCL F BIT(8) BASED(PT);
```

The following example will use GETITEM services to obtain the address, specifications and data for a list of five items from the same array. It will change the data and use PUTITEM services to update the array.

```
  ITEMLIST(1).NAME = 'B01  ';               /* BUILD LIST OF */
  ITEMLIST(2).NAME = 'B03  ';               /* ITEM NAMES */
  ITEMLIST(3).NAME = 'B05  ';
  ITEMLIST(4).NAME = 'B07  ';
  ITEMLIST(5).NAME = 'B09  ';
  P = ADDR(ITEMLIST(6).NAME);
  Q = NULL;                                 /* TERMINATE LIST */
  ITEMSTR.NAME = ADDR(ITEMLIST(1).NAME);    /* BUILD PARM LIST */
  ITEMSTR.NAMEINCR = 16;                    /* TO LOCATE ITEMS */
  ITEMSTR.AREA = ADDR(ITEMLIST(1).ADR);
  ITEMSTR.AREAINCR = 16;
  ITEMSTR.TYPE = 131;
  CALL DPPPIF(ITEMSTR.MACID);
/* ITEM ADDRESSES ARE RESOLVED  */
  DO I = 1 TO 5;                            /* ZERO UPPER BYTE */
  PT = ADDR(ITEMLIST(I).ADR);               /* OF ADDRESS WORDS */
  F = '00000000'B;
  END;
/* PREPARE PARM LIST TO GET ITEM SPECS  */
  ITEMSTR.AREA = ADDR(ITEMLIST(1).LNG);
  ITEMSTR.TYPE = 133;
  CALL DPPPIF(ITEMSTR.MACID);
/* ITEM SPECIFICATIONS OBTAINED    */
  ITEMSTR.NAME = ADDR(ITEMLIST(1).ADR);     /* BUILD */
  ITEMLIST(6).ADR = NULL;                   /* PARAMETER LIST */
  ITEMSTR.AREA = ADDR(ITEM(1));             /* TO READ */
  ITEMSTR.AREAINCR = 16;                    /* BY ADDRESS */
  ITEMSTR.TYPE = 144;
  CALL DPPPIF(ITEMSTR.MACID);
/* DATA HAS BEEN READ */
  DO I = 1 TO 5;
  ITEM(I) = 'THIS BLOCKS GONE';             /* ALTER DATA */
  END;
/* WRITE ARRAY UPDATES BY ADDRESS */
  ITEMSTR.TYPE = 192;
  CALL DPPPIF(ITEMSTR.MACID);
/* UPDATE IS COMPLETE */
```

PL/I GETBLOCK/PUTBLOCK Interface

This PL/I interface provides the programmer the facilities of the Special Real Time Operating System GETBLOCK and PUTBLOCK services. The default structure, BLOCKSTR (defined below), may be copied into the PL/I program via a %INCLUDE BLOCKDEF.

```
 DCL   1 BLOCKSTR,                     /* GET/PUT BLOCK STRUCTURE */
       2 MACID FIXED BIN INIT( 24),    /* BLOCK MACRO ID */
       2 RC FIXED BIN INIT(0),         /* RETURN CODE */
       2 NAME POINTER,                 /* A(NAMELIST/NUMBERLIST) */
       2 AREA POINTER,                 /* A(DATA ADDR-BLK NO. LIST) */
       2 ADD FIXED BIN INIT(8),        /* DATA AREA INCREMENT */
       2 TYPE FIXED BIN INIT(4);       /* TYPE OF BLOCK SERVICE */
```

BLOCKSTR
Name of the default structure provided for the Special Real Time Operating System blocked arrays service requests.

MACID
A halfword binary value of 24 identifying the service required to the interface routine.

RC
A halfword field containing a binary number return code from the

blocked array service routine. See GETBLOCK and PUTBLOCK macro write-ups for possible values.

NAME
The address of one of the following based on the specifications implied by TYPE.

a. If TYPE specifies 'NAME LIST', then NAME points to a list of 8-character array names followed by a X'FF' in the first byte after the last name where the next name would start.

ARRAY NAME LIST

0	NAME	
8	NAME	
16	FF	

b. If TYPE specifies 'NUMBER LIST', then NAME points to a list of halfword (2-byte) binary array numbers followed by a X'FF' in the first byte after the last number where the next number would start.

NUMBER LIST

0	NUMBER	
2	NUMBER	
4	FF	

AREA
The address of a list of 6-byte entries. ADD contains the value to be added to the list address to locate the next entry.

DATA AREA LIST

0	FLG	DATA AREA	BLK. NO.
6	FLG	DATA AREA	BLK. NO.
12	FLG	DATA AREA	BLK. NO.

FLG
A 1-byte flag field. A X'40' indicates the last data area and block number for a specified array, but not the end of the list. A X'80' indicates the last entry for the last array and the end of the list. A X'00' should appear in all other entries.

DATA AREA
A 3-byte address of the area into or from which the specified array block is moved.

BLK. NO.
A halfword binary number specifying the array block being moved.

ADD
A halfword binary value added to the contents of AREA to locate the next entry in the list. If zero, a length of 6 is assumed.

TYPE

A halfword binary value specifying the blocked array service options selected. The values (given in the tables below) identify the contents of NAME and whether it is a GETBLOCK with or without protection (PROTECT or RISK) or a PUTBLOCK.

NAME	AREA	PROTECTION REQUESTED	TYPE VALUE
A(NAME LIST)	A(DATA LIST)	RISK	4
A(NUMBER LIST	A(DATA LIST)	RISK	6
A(NAME LIST)	A(DATA LIST)	PROTECT	12
A(NUMBER LIST)	A(DATA LIST)	PROTECT	14

Figure 2-19. GETBLOCK Services

A(NAME LIST)	A(DATA LIST)	N/A	5
A(NUMBER LIST)	A(DATA LIST)	N/A	7

Figure 2-20. PUTBLOCK Services

The GETBLOCK/PUTBLOCK services are invoked in PL/I by CALLing DPPPIF with the properly completed structure (BLOCKSTR or a similar structure), the array name or number list and data address list.

The following example will GETBLOCK for block 5 from the two arrays BLK1 and BLOKB, and PUTBLOCK the block 5 of array BLK1 to block 5 of array BLOKB.

```
  DCL   1 BLOCKSTR,
        2 MACID FIXED BIN INIT(24),
        2 RC FIXED BIN INIT(0),
        2 NAME POINTER,
        2 AREA POINTER,
        2 ADD FIXED BIN INIT(8),
        2 TYPE FIXED BIN INIT(4);
  DCL   1 BLK,
        2 NAME (2) CHAR(8),
        2 NO (2) FIXED BIN,
        2 LIST (2),
            3 AREA POINTER,
            3 NUM FIXED BIN,
            3 RES FIXED BIN;
  DCL BLOCK (2) CHAR(256);
  DCL Q POINTER BASED(P);
  DCL F BIT(8) BASED(PT);
  BLK.NAME(1) = 'BLK1 ';
  BLK.NAME(2) = 'BLOKB ';
  BLK.NO(1) = -1;
  BLK.LIST(1).AREA = ADDR(BLOCK(1));
  PT = ADDR(BLK.LIST(1).AREA);
  F = '01000000'B;
  BLK.LIST(1).NUM = 5;
  BLK.LIST(2).AREA = ADDR(BLOCK(2));
  PT = ADDR(BLK.LIST(2).AREA);
  F = '10000000';
  BLK.LIST(2).NUM = 5;
  BLOCKSTR.NAME = ADDR(BLK.NAME(1));
  BLOCKSTR.AREA = ADDR(BLK.LIST(1).AREA);
  BLOCKSTR.TYPE = 12;
  CALL DPPPIF(BLOCKSTR.MACID);
/* BLOCK 5 ARRAYS BLK1 AND BLOKB HAVE BEEN READ */
  BLK.NAME(1) = BLK.NAME(2);
  P = ADDR(BLK.NAME(2));
  Q = NULL;
  PT = ADDR(BLK.LIST(1).AREA);
  F = '10000000'B;
  BLOCKSTR.TYPE = 5;
  CALL DPPPIF(BLOCKSTR.MACID);
/* BLOCK 5 OF ARRAY BLK1 HAS BEEN WRITTEN TO
  BLOCK 5 OF ARRAY BLOKB */
```

PL/I-GETLOG Interface

This PL/I interface provides the programmer the facilities of the GETLOG service. The default structure (defined below) may be copied into the PL/I program via a %INCLUDE GTLOGDEF;.

```
DCL 1 GTLOGSTR,                   /* GETLOG DEFAULT STRUCTURE */
    2 MACID FIXED BIN INIT(48),   /* GETLOG MACRO ID */
    2 RC    FIXED BIN INIT(0),    /* RETURN CODE */
    2 TYPE,                       /* PARAMETER LIST FLAGS */
       3 (F0,F1,                  /* RESERVED */
      LOGHDR,                     /* A(LOGHEADER) IN HEAD */
      F3,                         /* RESERVED */
      PROTECT,                    /* ON IF PROTECTION REQ'D */
      F5,                         /* RESERVED */
      NUMBER,                     /* NUMBERED ARRAY */
      F7)  BIT(1) INIT('0'B),     /* RESERVED */
    2 RES BIT(1) INIT('0'B),      /* RESERVED */
    2 NO FIXED BIN,               /* ARRAY NUMBER */
    2 AREA POINTER,               /* DATA AREA */
    2 STEP FIXED BIN(31,0) INIT(o), /* RELATIVE COPY NO */
    2 HEAD POINTER,               /* A(LOGHEADER/TIME FIELD) */
    2 NAME POINTER;               /* A(ARRAY NAME) */
```

GTLOGSTR
Name of the default structure provided for the Special Real Time Operating System GETLOG service requests.

MACID
A halfword binary value of 48 identifying the requested service to the interface routine.

RC
A halfword binary field containing a binary number return code from the GETLOG service routine. See GETLOG macro write-up for possible values.

TYPE
A flag's field indicating to the GETLOG service routine the options requested.

LOGHDR
If on, HEAD contains the address of a 24-byte log header identifying the relative starting point to determine which copy of the array will be retrieved from the log data set.

If off and HEAD is zero, the current copy becomes the relative starting point. If off and HEAD is not zero, then it contains the address of a 6-byte time and day field beginning on a fullword boundary. The first four bytes will contain a time in 10 millisecond units. The last two bytes will contain a binary value from 1 to 366 representing the day of the year. This time and day will be used as a comparison value to establish a relative starting point to determine which copy of the array will be retrieved from the log data set.

PROTECT
If on, a lock is set to prevent other programs from modifying the data set while this GETLOG is in process. If off, the data is moved without regard to other programs which may be storing into the data set.

NUMBER
If on, specifies that NO contains an array number. If off, NAME contains the address of an 8-character array name padded on the right with blanks if needed.

NO
Specifies the number of a numbered array for which a logged copy of the array is to be retrieved.

AREA
Specifies the address of a user-allocated storage area where the logged copy of the array will be written upon retrieval from the log data set. This area must be large enough to hold the entire array and a logheader (24 bytes).

STEP
Is used to determine which copy of a logged array, relative to the HEAD parameter will be retrieved from the log data set. The value is a signed number which may be either positive, negative, or zero.

HEAD
Zero or the address of an array logging header or of a 6-byte time and day field. See LOGHDR, under TYPE above for discussion of the contents of HEAD.

NAME
The address of the name of a numed array for which a logged copy of the array is to be retrieved.

The GETLOG service is invoked in PL/I by CALLing DPPPIF with a properly completed GTLOGSTR or a similar structure.

The following example will execute a GETLOG for the previously logged copy of array B referenced from the current copy. Note that the structure into which the log copy is read provides space for the log header.

```
DCL 1 GTLOGSTR,                                /* GETLOG DEFAULT STRUCTURE */
    2 MACID FIXED BIN INIT(48),                /* GETLOG MACRO ID */
    2 RC    FIXED BIN INIT(0),                 /* RETURN CODE */
    2 TYPE,                                    /* PARAMETER LIST FLAGS */
      3 (F0,F1,                                /* RESERVED */
      LOGHDR.                                  /* A(LOGHEADER) IN HEAD */
      F3,                                      /* RESERVED */
      PROTECT,                                 /* ON IF PROTECTION REQ'D */
      F5,                                      /* RESERVED */
      NUMBER,                                  /* NUMBERED ARRAY */
      F7)  BIT(1) INIT('O'B),                  /* RESERVED */
    2 RES BIT(1) INIT ('O'B),                  /* RESERVED */
    2 NO FIXED BIN,                            /* ARRAY NUMBER */
    2 AREA POINTER,                            /* DATA AREA */
    2 STEP FIXED BIN(31,0) INIT(0),            /* RELATIVE COPY NO */
    2 HEAD POINTER,                            /* A(LOGHEADER/TIME FIELD */
    2 NAME POINTER;                            /* A(ARRAY NAME) */
DCL A CHAR(8) INIT('B');
DCL 1 LARAY,
 2 LOGHD (12) FIXED BIN,
 2 ARRAY (24) FIXED BIN(31,0);
GTLOGSTR.STEP=-1;
GTLOGSTR.AREA= ADDR(LARAY.LOGHD(1));
GTLOGSTR.NAME= ADDR(A);
CALL DPPPIF(GTLOGSTR.MACID);
```

PL/I PUTLOG Interface

This PL/I interface provides the programmer the facilities of the PUTLOG service. The default structure, PTLOGSTR (defined below), may be copied into the PL/I program via a %INCLUDE PTLOGDEF;.

```
DCL 1 PTLOGSTR,                       /* PUTLOG DEFAULT STRUCTURE */
    2 MACID FIXED BIN INIT(44),       /* PUTLOG MACRO ID */
    2 RC    FIXED BIN INIT(0),        /* RETURN CODE */
    2 NAME  POINTER,                  /* A(NAME/NUMBER/LIST) */
    2 HEAD  POINTER,                  /* A(LOGHEADER/BLOCKLIST) */
    2 TYPE,                           /* PARAMETER LIST FLAGS */
      3 (F0,F1,                       /* RESERVED */
      LOGHDR,                         /* A(LOGHEADER) IN HEAD */
      BLOCK,                          /* A(BLOCKLIST) IN HEAD */
      PROTECT,                        /* ON IF PROTECTION REQ'D */
      LIST,                           /* A(LIST FORM) IN NAME */
      NUMBER) BIT(1) INIT('0'B),      /* A(NUMBER) IN NAME */
      3 PUT BIT(1) INIT('1'B),        /* MUST BE ON */
  2 RES BIT(1) INIT('0'B),            /* RESERVED */
  2 BLKADD FIXED BIN INIT(0);         /* DISPLACEMENT NEXT BLKNO */
```

PTLOGSTR
Name of the default structure provided for the Special Real Time Operating System PUTLOG service requests.

MACID
A halfword binary value of 44 identifying the requested service to the interface routine.

RC
A halfword binary field containing a binary number return code from the PUTLOG service routine. See PUTLOG macro write-up for possible values.

NAME
The address of an array name, number or a list of array names or numbers. The flags LIST and NUMBER in the flag field TYPE define the contents of this field.

LIST = '0'B and NUMBER = '0'B

Specifies the address of a name of a named array from which data is to be logged.

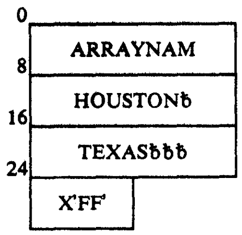
LIST = '0'B and NUMBER = '1'B

Specifies the number assigned to a numbered array from which data is to be logged in a halfword field binary field.

LIST = '1'B and NUMBER = '0'B

Specifies the address of a user-constructed list of array names from which data is to be logged. The name list will be a table of 8-byte entries with one valid array name in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the name list.

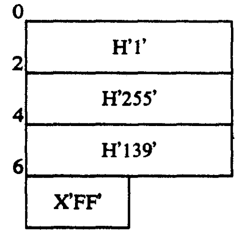
EXAMPLE: Name List



LIST = '1'B and NUMBER = '1'B

Specifies the address of a user-constructed list of array numbers from which data is to be logged. The number list will be a table of halfword entries with one valid array number in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the number list.

EXAMPLE: Number List



HEAD

The address of a logheader or blocklist or zero. The flags LOGHDR and BLOCK in the flag field TYPE define the contents of this field. If neither flag is set, HEAD is ignored.

LOGHDR = '1'B and BLOCK = '0'B

Specifies the address of an array logging header. Information in this logging header will identify the copy of the array which is to be replaced in the log data set.

The logging header is a 24-byte control block which precedes the array, both as the array exists in virtual storage and as is written to the logging array. The logging header which was retrieved as part of a previous GETLOG macro may be used to replace that copy in the log data set.

BLOCK = '1'B and LOGHDR = '0'B

Specifies the address of a user-constructed list of block numbers and of core addresses. The data list will be a table of 6-byte entries. Each entry will contain a 1-byte flag field, a 3-byte area address, and a 2-byte block number. This will allow the user to update selected segments of the DA log array for block VS resident arrays on demand basis. The latest log copy will be modified. However, the entire VS resident array is not logged; only the log block corresponding to the VS resident block specified will be updated. The actual log copy will not change; that is repeating PUTLOG macro calls with the BLOCK parameter will update the same log copy. A PUTLOG without the BLOCK parameter will cause the entire array to be logged to a new log copy.

PL/I BLKLIST Entry Description

0	1	2	3	4	5
FLAG BYTE	AREA ADDRESS			BLOCK NUMBER	

FLAG BYTE

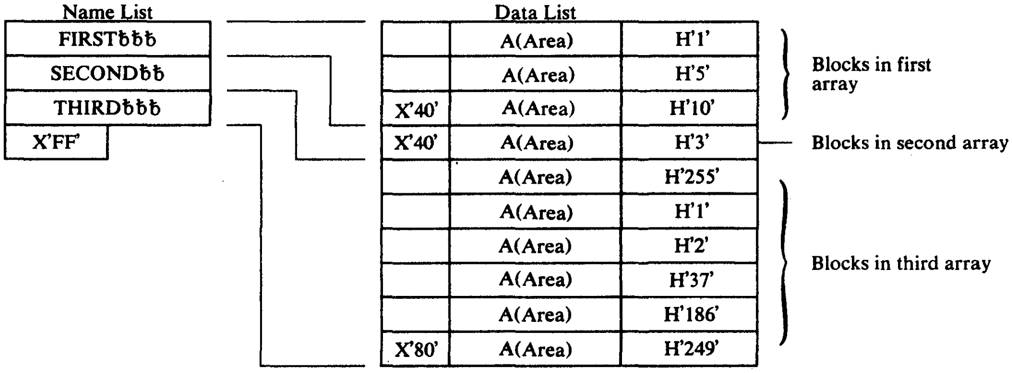
X'40' -- Indicates the last entry to be processed for a particular entry in the name list or number list.

X'80' -- Indicates the last entry in the data list.

AREA ADDRESS -- Ignored.

BLOCK NUMBER -- The number assigned to the data block to be retrieved and placed in the array described in the Name List or Number List.

EXAMPLE: BLKLIST and Name List



Name List
FIRSTbbb
SECONDbb
THIRDbbb
X'FF'

Data List		
	A(Area)	H'1'
	A(Area)	H'5'
X'40'	A(Area)	H'10'
X'40'	A(Area)	H'3'
	A(Area)	H'255'
	A(Area)	H'1'
	A(Area)	H'2'
	A(Area)	H'37'
	A(Area)	H'186'
X'80'	A(Area)	H'249'

TYPE -- A 1-byte flags field specifying the parameter options.

LOGHDR -- See HEAD.

BLOCK -- See HEAD.

PROTECT -- If on, a lock is set to prevent any other modifications to the data base during the PUTLOG service. If off, the data will be logged without regard to other concurrent modifications.

LIST -- See NAME.

NUMBER -- See NAME.

PUT -- Must be on for PUTLOG service.

BLKADD -- The value to be added to HEAD to locate the next block number. A value must be specified.

The PUTLOG service is invoked in PL/I by CALLing DPPPIF with a properly completed array name, array number, array name list, array number list, logheader address block list address and structure (PTLOGSTR or a similar structure).

The following example will PUTLOG Array B.

```
DCL 1 PTLOGSTR,                     /* PUTLOG DEFAULT STRUCTURE */
    2 MACID FIXED BIN INIT(44),     /* PUTLOG MACRO ID */
    2 RC    FIXED BIN INIT(0),      /* RETURN CODE */
    2 NAME  POINTER,                /* A(NAME/NUMBER/LIST) */
    2 HEAD  POINTER,                /* A(LOGHEADER/BLOCKLIST) */
    2 TYPE,                         /* PARAMETER LIST FLAGS */
       3 (F0,F1,                    /* RESERVED */
       LOGHDR,                      /* A(LOGHEADER) IN HEAD */
       BLOCK,                       /* A(BLOCKLIST) IN HEAD */
       PROTECT,                     /* ON IF PROTECTION REQ'D */
       LIST,                        /* A(LIST FORM) IN NAME */
       NUMBER) BIT(1) INIT('0'B),   /* A(NUMBER) IN NAME */
       3 PUT BIT(1) INIT('1'B),     /* MUST BE ON */
    2 RES BIT(1) INIT('0'B),        /* RESERVED */
    2 BLKADD FIXED BIN INIT(0);     /* DISPLACEMENT NEXT BLKNO */
DCL A CHAR(8) INIT('B');
PTLOGSTR.NAME = ADDR(A);
CALL DPPPIF(PTLOGSTR,MACID);
```

Note that because HEAD was left zero, the array was logged at the current log copy plus 1.

PL/I DUMPLOG Interface

This PL/I interface provides the programmer the facilities of the DUMPLOG service. The default structure DPLOGSTR (defined below), may be copied into the PL/I program via a %INCLUDE DPLOGDEF;.

```
DCL 1 DPLOGSTR,                         /* DUMPLOG PARAMETER STRUCTURE */
    2 MACID FIXED BIN INIT(52),         /* DUMPLOG MACRO ID */
    2 RC    FIXED BIN INIT(0),          /* RETURN CODE */
    2 TYPE,                             /* SERVICE OPTIONS FLAGS */
       3 (F0,F1,F2,                     /* RESERVED */
       DISP,                            /* NEW - IF ON */
       F4,                              /* RESERVED */
       LIST,                            /* LIST OF NAMES/NUMBERS */
       NUMB,                            /* ARRAY NUMBER/NUMB.LIST */
       F7) BIT(1) INIT('0'B),           /* RESERVED */
    2 RES BIT(1),                       /* RESERVED */
    2 NO    FIXED BIN INIT(0),          /* ARRAY NUMBER */
    2 START POINTER,                    /* A(START TIME) */
    2 STOP POINTER,                     /* (STOP TIME) */
    2 AREA POINTER,                     /* A(USER DATA) */
    2 DDNAM CHAR(8) INIT('DUMPLOG '),   /* DEFAULT DDNAME */
    2 LIST POINTER;                     /* A(NAME/NUMBER LIST) */
```

DPLOGSTR
Name of the default structure provided for the Special Real Time Operating System DUMPLOG service requests.

MACID
A halfword binary value of 52 identifying the requested service to the interface routine.

RC
A halfword binary field containing a binary number return code from the DUMPLOG service routine. See DUMPLOG macro write-up for possible values.

TYPE
A flags field indicating the requested options to the DUMPLOG service routine.

DISP
Specifies whether the dumped copies are to be written at the beginning of the dump data set (DISP = '1'B;) or added to the existing dumped copies (DISP = '0'B;).

If the disposition parameter specified on the DD card statement for this data set is either OLD or SHR and the data set is empty, then the first DUMPLOG request must specify NEW (DISP='1'B;).

Specifying DISP='1'B; on subsequent DUMPLOG requests will position a direct access data set to record one and will cause a tape data set to force the EOV before the log copies are written.

LIST
If on, specifies a list of array names or numbers is pointed to by the LIST pointer variable.

NUMB
If on, specifies numbered array(s) to be processed by this request. Either NO contains the array number, or LIST contains the address of a number list.

NO
Specifies the halfword number assigned to a numbered array for which the log array is to be dumped. LIST bit in TYPE must be off.

START
Specifies the address of a 6-byte time and day field beginning on a fullword boundary. The first four bytes will contain a time in 10-millisecond units. The last two bytes will contain a binary value from 1 to 366 representing the day of the year. The logged copies of the array will be searched until a copy is found with a log time equal to or greater than the start time specified. If this parameter is omitted, dumping will commence with the oldest logged copy of the array.

STOP
Specifies the address of a 6-byte time and day field beginning on a fullword boundary. The first four bytes will contain a time in 10-millisecond units. The last two bytes will contain a binary value from 1 to 366 representing the day of the year. The logged copies of the array will be dumped until the most recently logged copy has been dumped or until a copy is dumped with a log time equal to or greater than the stop time specified. If this parameter is omitted, dumping will terminate when the most recently logged copy of the array has been dumped.

Note: The DUMPLOG routine will insert a byte of X'FF'into the first byte of the logging header of the last copy of each array dumped to the sequential data set. This function to indicate the end of the dump of each array to the user delog routine.

AREA
Specifies the address of a 256-byte area of user data to be contained in the dump header for each array on the sequential dump data set.

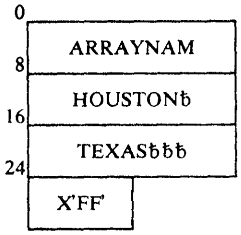
DDNAM
Specifies the name of a data definition statement which described a sequential data set to receive the dumped copies of the array from the log data set. If this parameter is omitted, the DD name 'DUMPLOG' will be assumed as the default.

The output will consist of spanned variable length records. The blocksize of the data set defined by the DDNAM parameter must be at least 264 bytes but no more than 32,760 bytes. The blocksize should be large enough to contain one array copy, the log header (24 bytes), the user dump header (256 bytes), if any, and the descriptor words for variable length records (8 bytes) for maximum processing efficiency.

LIST
Specifies the address of the array name of the log array to be dumped (LIST bit of TYPE and NUMB bit are off) or the address of a list of array names or numbers (LIST bit of TYPE is on).

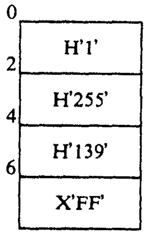
The name list will be a table of 8-byte entries with one valid array name in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the name list.

EXAMPLE: Name List



The number list will be a table of halfword entries with one valid array number in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the number list.

EXAMPLE: Number List



The DUMPLOG service is invoked in PL/I by CALLing DPPPIF with a properly completed DPLOGSTR or a similar structure.

The following example will DUMPLOG all the logged copies of array '13' beginning with the oldest copy. The dumped records will be at the start of the data set pointed to by DD name DUMPLOG.

```
DCL 1 DPLOGSTR,                        /* DUMPLOG PARAMETER STRUCTURE */
    2 MACID FIXED BIN INIT(52),        /* DUMPLOG MACRO ID */
    2 RC    FIXED BIN INIT(0),         /* RETURN CODE */
    2 TYPE,                            /* SERVICE OPTIONS FLAGS */
       3 (F0,F1,F2,                    /* RESERVED */
           DISP,                       /* NEW - IF ON */
           F4,                         /* RESERVED */
           LIST,                       /* LIST OF NAME/NUMBERS */
           NUMB,                       /* ARRAY NUMBER/NUMB.LIST */
           F7) BIT(1) INIT('0'B),      /* RESERVED */
    2 RES BIT(1),                      /* RESERVED */
    2 NO FIXED BIN INIT(0),            /* ARRAY NUMBER */
    2 START POINTER,                   /* A(START TIME) */
    2 STOP POINTER,                    /* A(STOP TIME) */
    2 AREA POINTER,                    /* A(USER DATA) */
    2 DDNAM CHAR(8) INIT('DUMPLOG'),   /* DEFAULT DDNAME */
    2 LIST POINTER;                    /* A (NAME/NUMBER LIST) */
DCL A CHAR(8) INIT ('B');
DPLOGSTR.TYPE.DISP = '1'B;
DPLOGSTR.LIST = ADDR(A);
CALL DPPPIF(DPLOGSTR.MACID);
```

Special Real Time Operating System FORTRAN Interfaces

This portion of the manual explains the programming considerations for FORTRAN programs to be run under Special Real Time Operating System environment. FORTRAN programs which do not use Special Real Time Operating System services should follow standard procedures as described in the *FORTRAN Programmer's Guide*, Form No. GC28-6817.

The remainder of this section explains procedures pertinent only if Special Real Time Operating System services will be used in FORTRAN programs. The user should be aware that these services are intended for FORTRAN programs which are invoked via the PATCH function. Other means of executing FORTRAN (such as LINK, CALL, XCTL) using these services should be used only by programmers who are aware of the interfaces between FORTRAN and the Special Real Time Operating System.

The interface routines described here use FORTRAN COMMON areas to pass and receive parameters. It should be noted that, when using the G level FORTRAN compiler, the variable name that is passed to the interface routine(s) must be the name of a variable within the COMMON area and not the name of the COMMON area.

Enhancements to FORTRAN Data Manipulation and Movement

This section describes three enhancements provided to the FORTRAN programmer to interface with Special Real Time Operating System:

1. Identification of the computer storage address of one variable and setting another variable to that value.

2. Execution of storage bits.

3. Movement of up to 32,767 computer storage bytes of data from one location to another.

The FORTRAN programmer will discover that one or more of these capabilities is probably needed when using the capabilities described in the remainder of this section.

IADDR Function

This function computes and returns to the caller the 32-bit address requested and stores it at the desired location.

```
X=IADDR(Y)
```

ORBIT Subroutine

This subroutine ORs the specified bit mask into the specified address. The location to be modified must be specified first in the CALL parameters.

```
LOGICAL*1 FF/ZFF/
      .
      .
      .
CALL ORBIT(X,FF)
```

NDBIT Subroutine .

This subroutine ANDs the specified mask with data at the specified address. The location to be modified must be specified first in the call parameters.

```
 LOGICAL*1 SF/Z7F/
       .
       .
       .
 CALL NDBIT(X,SF)
```

COPY Subroutine

This function moves up to 32,767 contiguous bytes of data from one location to another. More than one move operation can be specified in the same call. The format of the CALL subroutine is as follows:

CALL COPY(INLIST,OUT$_1$,OUT$_2$...,OUT$_n$)

where INLIST specifies an address variable or a common area consisting of address variables. (An address variable is one which has been set using the IADDR function to contain the address of, or point to, a data area.) The address variable(s) in INLIST should point to the data area from which data is to be moved. The variable OUT$_1$,...,OUT$_n$ should be the labels for the data area to which the data is to be moved. The number of copy operations will be equal to the number of OUT$_n$ variables. Data will be moved to the OUT$_1$ data area from the data area addressed by the first address variable of INLIST, data will be moved to the OUT$_2$ data area from the data area addressed by the second address variable of INLIST, and so on until OUT$_n$ has been likewise processed.

If either OUT$_i$ or the corresponding address variable of inlist is zero no move takes place for that OUT$_i$. The copy operation proceeds to the next OUT$_{i+1}$.

The length of each move will be determined independently for each OUT$_i$ either by the first halfword of the OUT$_i$ data area, or by the first halfword of the data area pointed to by the corresponding address variable of INLIST. The length of the move will be equal to the smaller positive value of the two halfwords, as zero and negative values are not recognized. (No move will take place for this OUT$_i$ if neither halfword referred to is positive.) Note that the first halfword of both areas is included in the move, and the two bytes occupied by the length must be included in the length specification.

For example, moving one data area, INA1, to another data area, OUTA1, is accomplished as follows:

Given:

```
COMMON/INA1/INHALF, INF (30)
INTEGER INHALF*2, INF*2
COMMON/OUTA1/OUTHAF, OUTF (70)
INTEGER OUTHAF*2, OUTF*2
```

Then code:

INHALF=30*2+2	Set the first halfword of the input area equal to its length.
OUTHAF=70*2+2	Set the first halfword of the output area equal to its maximum length - in case the input area's length varies.

```
 INADD=IADDR(INHALF)        Set the address variable INADD to point to
                            labeled common area INA1.

 CALL COPY(INADD,OUTHAF)
```

This causes the common area INA1 to be moved in its entirety (62 bytes) to the common area OUTA1. Note that the value of OUTHAF would be 62 after the MOVE operation.

The next example describes moving several common areas (COM1, COM2, COM3) to output common areas (OUT1, OUT2, COUT3). It is assumed that the first halfword of each common area has been set to its desired length.

Given:

```
 COMMON/COM1/CHALF1...
 COMMON/COM2/CHALF2...
 COMMON/COM3/CHALF3...
 COMMON/INCOM/IN1,IN2,IN3
 INTEGER  IN1,IN2,IN3
 COMMON /OUT1/OHALF1...
 COMMON /OUT2/OHALF2...
 COMMON /OUT3/OHALF3...
```

Then code:

```
 IN1=IADDR(CHALF1)         Set the address variables
 IN2=IADDR(CHALF2)         to point to their common
 IN3=IADDR(CHALF3)         areas.
 CALL COPY(IN1,OHALF1,OHALF2,OHALF3)
```

Locating Input Parameter(s) After Being PATCHed

This section explains the coding of FORTRAN program to interrogate the data which may have been specified via the PATCH function. Refer to the section on the PATCH macro for a detailed discussion of the PATCH parameters. The only requirement of this discussion is to know that the PATCH macro can specify a list of input parameters to be passed to the user in his PROBL. This discussion applies to a program which is re-entered at the beginning for each execution of the function to be processed. A FORTRAN program may be coded to be logically entered several times when actually being entered at the beginning only once. This is described in the section entitled, "Repeated Execution of a FORTRAN Program".

The FORTRAN program receiving control due to the PATCH can gain access to this PROBL parameter list by including a call to a special interface routine (DPPFPM) and having a predefined common area properly initialized, as described. The common area is described in the following example and will be called PARM throughout this write-up.

PARM

	0	+2
	PRMAC	PRMRC
+4	PRXCVT	
+8	PRMRES	
+12	PRMADD	

While the first two halfwords, PRMAC and PRMRC, must be initialized to zero prior to calling DPPFPM, the remainder of the common area need not be initialized.

Following is a layout of the common area PARM in FORTRAN code, with an explanation of each variable as they pertain to this section.

```
C
C COMMON NAMED'PARM'--PARAMETER TABLE FOR RECEPTION OF PATCH
 COMMON/PARM.PRMAC
    INTEGER*2 PRMAC
 COMMON/PARM/PRMRC
    INTEGER*2 PRMRC
 COMMON/PARM/PRXCVT
    INTEGER*4 PRXCVT
 COMMON/PARM/PRMRES
    INTEGER*4 PRMRES
 COMMON/PARM/PRMADD
    INTEGER*4 PRMADD
C END OF COMMON NAMED 'PARM'
C
```

PRMAC

A halfword variable reserved for use by DPPFPM. It must be initialized to zero prior to calling DPPFPM (the first call only) and its contents will have no meaning to the FORTRAN programmer. It should not be altered after the first call to DPPFPM.

PRMRC

A halfword variable which will contain the return code as set by DPPFPM. This variable should be set to zero prior to calling DPPFPM. Its subsequent contents have no meaning to the FORTRAN programmer when calling DPPFPM solely to gain access to the PATCHed input parameters. (The section entitled, "Repeated Execution of FORTRAN Program" describes another use of the program DPPFPM and common area PARM in which this variable is pertinent).

PRXCVT

This fullword variable, which need not be initialized, contains the address of the XCVT after calling DPPFPM. This variable does not pertain to the present discussion.

PRMRES

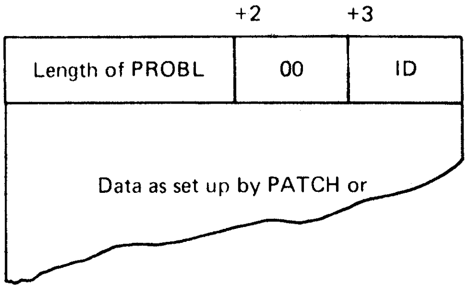
This fullword variable, which need not be initialized, contains the address of the Resource Table for this task after calling DPPFPM. This variable is not pertinent to the present discussion on input parameters.

PRMADD

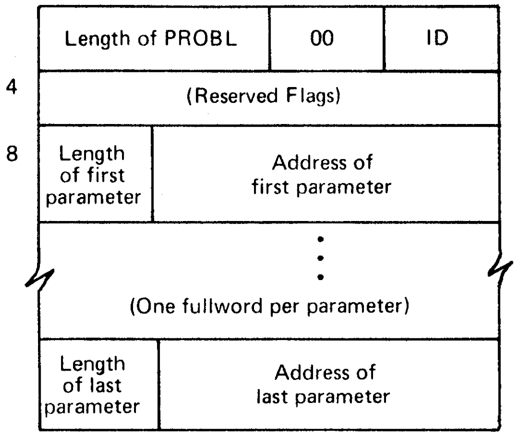
This fullword variable, which need not be initialized contains the address of the Problem Parameter List (PROBL) for the causative PATCH. It is through this variable that the FORTRAN programmer gains access to his PROBL, which contains the input parameters or pointers to the input parameters.

The PROBL, pointed to by the variable PRMADD (within common PARM), should also be described by a labeled common area, which shall henceforth be called PROBL. The PROBL has one of two formats, depending on whether this program is PATCHed via a PATCH macro (from an already executing program) or via an input control stream PATCH CARD. In either case, the length of the table varies with the number of parameters passed on the PATCH, so the common area (PROBL) should be specified according to the maximum number of parameters expected. The following example depicts the two formats of the PROBL.

PROBL from
PATCH Macro



PROBL from
an Input Control Stream PATCH Card


Length of PROBL
00
ID
4
(Reserved Flags)
8
Length of first parameter
Address of first parameter
(One fullword per parameter)
Length of last parameter
Address of last parameter

Note that the first word of both formats is the same and that while the format of the remainder is fixed in the PATCH card type, the format of the remainder is flexible in the PATCH macro type. In most cases the PROBL of a PATCH macro, when used in conjunction with FORTRAN programs, will be set up to consist of data rather than pointers to data as in the PATCH card type.

The common area, then, for the PROBL would be coded as follows:

```
 COMMON/PROBL/PRBLNG
    INTEGER*2 PROBLNG
 COMMON/PROBL/ID
    INTEGER*2 ID
 COMMON/PROBL/PROBP1
    INTEGER*4 PROBP1
```

PRBLNG
The total length in bytes of this PROBL, including this halfword.

ID
The ID value specified on the PATCH CARD or macro (defaults to zero).

PROBP1
A fullword variable which (1) contains the first PATCH parameter or (2) contains the address of the first PATCH parameter. (1) or (2) is the user's (caller's) option.

The Special Real Time Operating System initialization process allows a PATCH to be executed under control of a PATCH card. The format of the parameters that may be specified with the PATCH card are such that passing parameters from the PATCH card may not be practical, without an assembler language routine specially written for this purpose.

In the following example, assume that the FORTRAN program will be PATCHed by a program (already in execution) with which a common format for the PROBL has been previously established. Suppose that the following statements describe this PROBL format and appear in the FORTRAN program.

```
  COMMON/PROBL/PRBLNG,ID,PROBP(10)
  INTEGER PRBLNG*2,ID*2,PROBP*4
```

These cards indicate that 10 fullword parameters will be passed to this FORTRAN program.

Ther, to interrogate the parameters within the FORTRAN program, code the following:

```
  COMMON/PARM/PRMAC,PRMRC,PRXCT,PRMRES,PRMADD
  INTEGER PRMAC*2,PRMRC*2,PRXCVT,PRMRES,PRMADD
```

These cards defined the common area PARM previously explained.

PRMAC=0
PRMC=0
	These statements initialize the common area PARM as required.

CALL DPPFPM(PRMAC)	Cause the common area PARM to be properly filled in.
PRBLNG=44	Set length of PROBL to maximum expected.
CALL COPY(PRMADD,PRBLNG)	Cause the common area PROBL to be filled in as per the PROBL of the PATCHING program.

The variables PROBP(1) through PROBP(10) will have the values specified in the PATCH and can be referenced normally.

Repeated Executions of a FORTRAN Program

This section describes the procedure to be used for a FORTRAN program which is to be repeatedly executed in a realtime environment.

Under standard executing conditions when a FORTRAN program is executed a second time after completing one execution, a fresh copy is fetched and both prologue and epilogue are executed again. This fetching of a fresh copy and the re-executing of the prologue/epilogue can sometimes be avoided when executing a FORTRAN program under the Special Real Time Operating System. This is done by coding multiple calls to an interface program (DPPFPM) in a certain sequence and by having a predefined common area properly initialized.

The description of the common area, hereafter called PARM, was presented in the section entitled "Locating Input Parameters after being PATCHed" and will be repeated here with explanations pertinent to this section. The following example depicts the PARM common area.

PARM

	0	+2
	PRMAC	PRMRC
+4	PRXCVT	
+8	PRMRES	
+12	PRMADD	

The FORTRAN-coded statements for the specification of the PARM common area follows:

```
C
C COMMON NAMED'PARM'--PARAMETER TABLE FOR RECEPTION OF PATCH
 COMMON/PARM/PRMAC
    INTEGER*2 PRMAC
 COMMON/PARM/PRMRC
    INTEGER*2 PRMRC
 COMMON/PARM/PRXCVT
    INTEGER*4 PRXCVT
 COMMON/PARM/PRMRES
    INTEGER*4 PRMRES
 COMMON/PARM/PRMADD
    INTEGER*4 PRMADD
C END OF COMMON NAMED 'PARM'
C
```

PRMAC
This halfword variable should be initialized to zero prior to the first call to DPPFPM only. It will be used by DPPFPM and should not be subsequently altered by this FORTRAN program. Its contents will be of no significance to the FORTRAN programmer.

PRMRC
This halfword variable should be initialized to zero prior to the first call to DPPFPM only. It is used as both an input variable and an output variable by the FORTRAN program following An in-depth explanation of the use of this vital parameter follows the remaining description of the PARM common area.

PRXCVT
This fullword variable, which need not be initialized, contains the address of the XCVT. This variable is immaterial to our present discussion.

PRMRES
This fullword variable, which need not be initialized, contains the address of this task's resource table. This variable is immaterial to our present discussion.

PRMADD
This fullword variable, which need not be initialized, contains the address of the PROBL (problem parameter list) for this PATCH. Refer to the section entitled "Locating Input Parameters after being PATCHed" for a full explanation of accessing the problem parameter list.

To understand the use of the PARM common area (in particular the variable PRMRC) in conjunction with multiple calls to DPPFPM, the concept of a work queue for a task must be understood. The FORTRAN program should be capable of performing a specified function when invoked, and if this function is requested again (or several times) under the same task, then the second request will be the first entry in the work queue of that task.

When the original request is serviced and the FORTRAN program has returned, the second request (first entry in the work queue) will be honored and the FORTRAN program will be executed again. When the FORTRAN program has returned and no additional requests are waiting, a condition indicated by an empty work queue, the Special Real Time Operating System will place this task in wait state until such a request is made. If a request is made for a different program to be executed under this task, the Special Real Time Operating System will allow this FORTRAN program to be purged.

If the author of the FORTRAN program foresees no entry in the work queue for this program's task at the completion of the program or at a later time, he should return as in standard FORTRAN (i.e., he need not call DPPFPM at all except for input parameter considerations).

Given that entries in this task's work queue for this program are expected on completion of the processing of this PATCH, the programmer can avoid the overhead of program fetch and epilogue/prologue by not returning normally but instead, calling DPPFPM again. (The first call to DPPFPM must have been done). Through the use of the variable PRMAC (maintained by DPPFPM), DPPFPM will know that this is not the first call and consider it a RETURN. DPPFPM will then locate the first entry in the work queue for this task (or wait until there is one) and, if the entry is for this FORTRAN program, properly fill in the PARM common area according to the PATCH causing this work queue entry.

If the previous PATCH (the one for which processing has just completed) had specified an ECB for posting, then that ECB will be posted with a completion code equal to the value in the variable PRMRC, which can be set by this program. If the first entry in the work queue was for another program, then this FORTRAN program should return normally, yielding this task to the new request. This This condition will be indicated by a non-zero value in PRMRC. This setting of PRMRC by DPPFPM is done on each call, including the first call, so the FORTRAN coder can surmise that only one call to DPPFPM is needed, followed by a RETURN if PRMRC is not zero. At the end of processing, or anywhere a normal RETURN would be coded, he would GO TO the statement of the CALL to DPPFPM.

The following example illustrates the use of the multiple calls to DPPFPM for a FORTRAN program written with the expectation that it would be PATCHed repeatedly under the same task.

Given:

A FORTRAN program that expects to be PATCHed (via PATCH macro) with different PATCH IDs to indicate various macro processing options desired.

Then code:

```
COMMON/PARM/PRMAC,PRMRC,PRXCVT,PRMRES,PRMADD
INTEGER PRMAC*2,PRMRC*2,PRXCVT,PRMRES,PRMADD
COMMON/PROBL/PRBLNG,ID,PROBP1
INTEGER PRBLNG*2,ID*2,PROBP1
```

These cards define the required common areas.

	PRMAC=0
	PRMC=0
		Initialize the PARM common area as required

1000	CALL DPPFPM(PRMAC)	Call DPPFPM to get PATCH ID parameters
	IF(PRMRC.NE.0) RETURN	Non-zero indicates a different program has been PATCHed to execute under this task. Return and give up control of this task. <u>This condition will not occur on the first call</u>.

A zero value indicates that variables in the PARM common area (especially PRMADD) have been set according to the PATCH. Proceed with processing.

```
PRBLNG=12                 Set PRBLNG for a copy operation.

CALL COPY(PRMADD,PRBLNG)  Copy the PROBL for this PATCH to the common
                          area PROBL.
```

Now inspect ID and proceed processing.

When processing is completed, set PRMRC according to a previously agreed upon return code (agreed upon with the author of the program which executed the PATCH this program has been processing).

```
PRMRD= return code

GO TO 1000          This causes another call to DPPFPM indicating that
                    processing is completed for this PATCH and the
                    program expects another.
```

Note: The input parameters of each PATCH could also be interrogated in this example. Refer to the previous section for a description of this procedure.

Special Real Time Operating System Online Macro Subroutines for FORTRAN

This section explains the coding of each of the online macros provided to the FORTRAN programmer by the Special Real Time Operating System. Each of these functions is provided to assembler language programmers through macro calls. There is a parallel (but more detailed) write-up on each function in the online macro section of this manual. Although this section may attempt to explain to varying degrees the functions themselves, the main purpose here is to describe the format of the COMMON areas required for invoking each function and point out peculiarities where pertinent.

FORTRAN-PATCH Interface

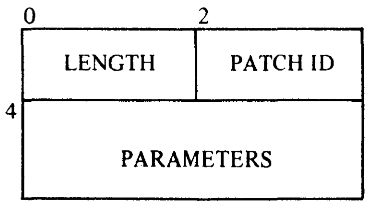
The PATCH service provides the programmer the facility of creating work queues for passing parameters to programs executing under the Special Real Time Operating System. The following FORTRAN statements define the parameter list for this service:

```
C
C COMMON NAMED 'PATCH'--PARAMETERS NECESSARY FOR PATCH FROM
C FORTRAN
C
 COMMON/PATCH/PATMAC
    INTEGER*2 PATMAC
 COMMON/PATCH/PATRC
    INTEGER*2 PATRC
 COMMON/PATCH/PATPRM
    INTEGER*4 PATPRM
 COMMON/PATCH/PATASK
    INTEGER*4 PATASK(8)
 COMMON/PATCH/PATEP
    LOGICAL*1 PATEP(8)
 COMMON/PATCH/PATNAM
    LOGICAL*1 PATNAM(8)
 COMMON/PATCH/PATQ
    INTEGER*2 PATQ
 COMMON/PATCH/PATV
    INTEGER*2 PATV
 COMMON/PATCH/PATECB
    INTEGER*4 PATECB
 COMMON/PATCH/PATRES
    INTEGER*4 PATRES(2)
 COMMON/PATCH/PATCBX
    INTEGER*4 PATCBX
 COMMON/PATCH/PATFLG
    LOGICAL*1 PATFLG
C END OF COMMON NAMED 'PATCH'
C
```

PATMAC
A halfword binary constant value of zero to identify a PATCH service request to the interface routine.

PATRC
A halfword binary field containing the return code from the service routine. See PATCH macro write-up for possible values.

PATPRM
A fullword address of the parameter list being passed. The format is a halfword binary value (minimum value is 4) describing the length of the entire parameter list, (including length and patch ID) followed by a halfword binary value from 0 to 255 called the PATCH ID with the remainder of the list being the parameters. The diagram below represents the format of a PATCH problem parameter list.



PATASK
An 8-byte character field containing the name of the task being PATCHed. If the task does not exist, one by that name will be created. If PATASK is all blanks, the PATCHed program will execute under a dependent task.

PATEP
An 8-byte character field containing a valid entry point name which

is the name of the program to be scheduled under the task being created with the PATCH.

PATNAM and PATV
Specifies an 8-byte character field containing the task name for determining priority and a halfword binary value which will determine that priority relative to the task name in PATNAM.

PATQ
A halfword binary value from 0 to 235 specifying the number of work queue entries to be allowed for the new independent task. If 0 is specified, the task accepts one PATCH, works on that request, and, when completed, waits for the next request. If a PATCH is requested for that task while it is busy, the request is not executed. If the queue length is 1, the task can accept one PATCH even while it is busy. Any PATCH parameters waiting in the queue when a task completes processing of the current request will be executed one at a time, with the start of the queue being executed first. This procedure is the same for all queue values from 0 to 255.

PATECB
The address of a fullword event control block (ECB) within a PATCH-WAIT parameter list of the common area WAIT. The ECB is posted when processing for this PATCH completes.

PATRES
Filler position for required space for PATCH MACRO (not usable by programmers).

PATCBX
A fullword address of the TCB extension control block (TCBX) for an existing independent task. The TCBX address is stored by the interface routine after each PATCH service call. Use of this parameter with all successive PATCHes to the same independent task after the initial PATCH will reduce system processing time. Note that the other parameters must still be specified for verification or in the event the task has been DPATCHed.

PATFLG
The PATCH option flags as described below:

X'40' -- This PATCH is intended for the MASTER partition.

X'20' -- This PATCH is intended for the SLAVE partition.

X'08' -- If this work request is pushed off the queue, the ECB is to be posted with a REPATCH control block address.

X'04' -- Place the work request at the start of the work queue. If off, the request is queued last.

X'02' -- Place this work request on the task DPATCH queue to be executed when a DPATCH is issued for this task.

X'01' -- Specifies a DELETE is to be issued for the load module named previously after processing completes for this PATCH.

X'00' -- Execute this PATCH last.

All combinations are valid except X'04' and X'02' must not both be set to 1.

The PATCH service may be invoked by assigning values to the above defined variables and CALLing DPPPIF passing the common area as the (only) parameter. Examples of using the PATCH facility follow.

Examples 1 and 2 use the following parameter lists, variables, and constants as expressed in FORTRAN statements:

```
    BLOCK DATA
      COMMON/PATCH/PATMAC,PATRC,PATPRM,PATASK(2),PATEP(2),PATNAM(2),
     1PATQ,PATV,PATECB,PATRES(2),PATCBX,PATFLG
      INTEGER PATMAC*2/0/,PATRC*2/0/,PATPRM,PATQ*2/1/,PATV*2/0/,
     1PATECB,PATRES,PATCBX
      LOGICAL PATASK*4/'    '/,'    ',PATEP*4,PATNAM*4/'    /,'    ',
     1PATFLG*1
      COMMON/WAIT/WTMAC,WTRC,WTECB
      INTEGER WTMAC*2/60/,WTRC*2/0/,WTECB/0/
      END
```

(The above common areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
      LOGICAL*4 TN(2)/'DPPZ','TSOO'/,TP(2)/'DPPZ','TS13'/
      LOGICAL*4 DP(2)/'DEPE','NDX'/,BLK(2)/'    ','    '/
```

Example 1

In this example, the task DPPZTSOO is to be created with a queue length of 1. Program DPPSTS13 is to be executed, and the parameter list is to contain only the length field and a PATCH ID of 10. The new task is to have the same priority as the task issuing the PATCH. Note that if the task already exists, the PATFLG (all bits off) indicates this work request will be queued behind any others on the queue.

```
    PRBLNG=4
      ID=10
      PATPRM=IADDR(PRBLNG)
      DO 100 I=1,2
      PATASK(I)=TN(I)
  100 PATEP(I)=TP(I)
      CALL DPPPIF(PATMAC)
```

Example 2

In this example, assume that the CALL in Example 1 has returned, and a dependent task is to be created at a priority of 10 less than the task DPPZTSOO and that program DEPENDX is to be passed a parameter list PATCH ID of 2. The PATCHing program will WAIT for the dependent task to complete. The WAIT function is executed via a CALL to the interface routine using the WAITSTR structure.

```
   CALL DPPPIF(PATMAC)
      ID=2
      DO 200 I=1,2
      PATASK(I)=BLK(I)
      PATEP(I)=DP(I)
  200 PATNAM(I)=TN(I)
      PATV=10
      PATECB=IADDR(WTECB)
      CALL DPPPIF(PATMAC)
      IF(PATRC.GE.8) GO TO 400
      CALL DPPPIF(WTMAC)
  400 CONTINUE
```

FORTRAN PATCH-WAIT Interface

This interface provides the FORTRAN programmer with the facility to wait for the completion of a WQE generated by a PATCH. The following FORTRAN statements define the interface parameter list:

```
C
C COMMON NAMED 'WAIT'--PARAMETER TABLE FOR WAIT
    COMMON/WAIT/WTMAC
    INTEGER*2 WTMAC
    COMMON/WAIT/WTRC
    INTEGER*2 WTRC
    COMMON/WAIT/WTECB
    INTEGER*4 WTECB
C END OF COMMON NAMED 'WAIT'
C
```

WTMAC
A halfword binary constant value of 60 identifying the requested service to the interface routine.

WTRC
A halfword binary number containing the high order byte of the completion code from the PATCHed program. See PATCH macro for possible values. It should be initialized to zero.

WTECB
A fullword binary field containing the 3 low order bytes of the completion code from the WQE just processed or the address of a REPATCH control block. The value of this field is governed by the contents of WTRC. It should be initialized to zero.

Note: For this interface, WTRC will never be zero when the interface returns to the FORTRAN program.

Example 2 of the FORTRAN-PATCH interface shows the correct method for using this service.

FORTRAN-DPATCH Interface

The DPATCH facility provides the programmer the method of destroying tasks which were created by the PATCH service. The following FORTRAN statements define the parameter list for this service:

```
C
C COMMON NAMED 'DPATCH'
    COMMON/DPATCH/DPRES
    INTEGER*2 DPRES
    COMMON/DPATCH/DPMAC
    INTEGER*2 DPMAC
    COMMON/DPATCH/DPRC
    INTEGER*2 DPRC
    COMMON/DPATCH/DPTYP
    INTEGER*2 DPTYP
    COMMON/DPATCH/DPTSK
    LOGICAL*1 DPTSK(8)
 C END OF COMMON NAMED 'DPATCH'
 C
```

DPRES
A halfword field inserted to align DEPTSK on a fullword boundary.

DPMAC
A halfword binary constant value of 8 identifying to the interface routine the required service.

DPRC
A halfword binary field containing a binary number return code from the service routine. See DPATCH macro write-up for return codes. It should be initialized to zero.

DPTYP
A halfword binary value specifying the DPATCH service requested. If 0 is specified, the task is deleted immediately or when the currently executing work request completes. Any work queued to the task is posted as deleted. If 4 is specified, the task is deleted only if its work queue is empty. This does not prevent new work from being queued. If 12 is specified, the task is deleted even if it is active. See the DEPATCH function under ONLINE MACRO for further explanation of the DEPTYP operand in the DEPATCH function.

DPTSK
Two logical fullwords specifying the name of the task being deleted. If blank, the current task is deleted. If the task is active, the program that is running will be ABENDed.

The following example will force the task named 'BOLDTASK' to be DPATCHed immediately regardless of its active state and the amount of queued work. If the task is active, the running program will be ABENDed.

```
C  FORTRAN DEPATCH EXAMPLE
      BLOCK DATA
       COMMON/DEPTCH/DEPRES,DEPMAC,DEPRC,DEPTYP,DEPTSK(2)
       INTEGER DEPMAC*2/8/,DEPRC*2/0/,DEPTYP*2/0/,DEPRES*2
       LOGICAL DEPTSK*4
   END
```

(The above common areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
       LOGICAL A*4(2)/'BOLD','TASK'/
             .
             .
             .
       DEPTSK(1)=A(1)
       DEPTSK(2)=A(2)
       DEPTYP=12
       CALL DPPPIF(DEPMAC)
```

<u>FORTRAN-REPATCH</u> <u>Interface</u>

This FORTRAN interface provides the programmer the facilities of the Special Real Time Operating System REPATCH service. The following FORTRAN statements define the parameter list for this service:

```
C
C COMMON NAMED 'RPATCH'--PARAMETER TABLE FOR RPATCH
 COMMON/RPATCH/RPMAC
    INTEGER*2 RPMAC
 COMMON/RPATCH/RPRC
    INTEGER*2 RPRC
 COMMON/RPATCH/RPTYP
    INTEGER*4 RPTYP
 COMMON/RPATCH/RPCB
    INTEGER*4 RPCB
 COMMON/RPATCH/RPTSK
    LOGICAL*1 RPTSK(8)
 COMMON/RPATCH/RPEP
    LOGICAL*1 RPEP(8)
 COMMON/RPATCH/RPRTK
    LOGICAL*1 RPRTK(8)
 COMMON/RPATCH/RPQUE
    INTEGER*2 RPQUE
 COMMON/RPATCH/RPVAL
    INTEGER*2 RPVAL
 COMMON/RPATCH/RPECB
    INTEGER*4 RPECB
 COMMON/RPATCH/RPRES
    INTEGER*4 RPRES(2)
 COMMON/RPATCH/RPTCB
    INTEGER*4 RPTCB
 COMMON/RPATCH/RPFLG
    LOGICAL*1 RPFLG
 COMMON/RPATCH/RPAD
    LOGICAL*1 RPAD(3)
 COMMON/RPATCH/RPPRM
    INTEGER*4 RPPRM(3)
C END OF COMMON NAMED 'RPATCH'
C
```

RPMAC
A halfword binary value of 12 identifying to the interface routine the required service.

RPRC
A halfword field containing a binary number return code from the REPATCH/PATCH service routine. See REPATCH macro write-up for REPATCH and related PATCH return codes.

RPTYP
A fullword binary value indicating the interface routine service required:

0 -- The REPATCH control block is to be copied to this parameter list for alteration prior to REPATCH.

4 -- Issue REPATCH TYPE = EXEC.

8 -- Issue REPATCH TYPE = PURGE.

RPCB
A fullword binary field to contain the REPATCH control block address placed in the WTECB when WTRC equals 68. The value in WTECB must be moved to RPCB before any interface call except the first interface call RPTYP = 4 or 8 following a RPTYP = 0 interface call.

RPTSK
Two 4-byte logical words containing the name of the task being referenced by this PATCH.

RPEP
Two 4-byte logical words containing the name of the program to be scheduled under task specified in RPTSK.

RPRTK and RPVAL
Specifies two 4-byte logical words containing a task name and a halfword value which will determine the priority of the new task relative to the named task in RPRTK.

RPQUE
A halfword specifying the number of work queue entries to be provided for a new independent task.

RPECB
Specifies the address of the ECB within a COMMON/WAIT area which is to be used in a CALL DPPPIF. This ECB is posted when processing for this PATCH completes. The ECB which contained the REPATCH control address may be reused and will be if this parameter is left unchanged.

RPRES
Filler position required by REPATCH macro (not used by programmer).

RPTCB
Contains the address of the TCB extension control block for an existing independent task.

RPFLG
The PATCH option flags as described below:

X'40' -- This PATCH is intended for the MASTER partition.

X'20' -- This PATCH is intended for the SLAVE partition.

X'08' -- If this work request is pushed off the queue, the ECB is to be posted with a REPATCH control block address.

X'04' -- Place the work request on the front of the work queue. If off, the request is queued last.

X'02' -- Place this work request on the task DPATCH queue to be executed when a DPATCH is issued for this task.

X'01' -- Specifies that a DELETE is to be issued for the load module named above after processing completes for this PATCH.

Codes X'04' and X'02' are mutually exclusive; all other combinations are allowed.

RPPAD, and RPPRM
Pointers which must not be altered by programmer.

The Special Real Time Operating System REPATCH service may be invoked by a FORTRAN program by defining a COMMON area as described above, moving the REPATCH control block address from the event control block to the RPCB field and then doing one of the following:

a. If a REPATCH is to be executed without changes, set RPTYP to 4 or 8 and CALL DPPPIF.

b. If the REPATCH is to be changed prior to execution, set RPTYP = 0, CALL DPPPIF, make changes desired, set RETYP to 4 and CALL DPPPIF again.

Users of this facility should be aware that only the supervisor portion of the PATCH parameters can be altered. The problem parameters cannot be changed. All REPATCH control blocks must be returned to the system through a RPTYP = 4 or 8 service request.

Examples 1 and 2 show the various methods of using REPATCH. The example for using REPATCH service in FORTRAN use the following definitions of COMMON areas and constants:

```
    BLOCK DATA
      COMMON/RPATCH/REPMAC,REPRC,REPTYP,TEPCB,REPTSK(2),
     1REPEP(2),REPRTK(2),REPQUE,REPVAL,REPECB,REPRES(2),
     1REPTCB,REPFLG,REPAD(3),REPPRM(3)
      LOGICAL REPTSK*4,REPEP*4,REPRTK*4,REPFLG*1,REPAD*1
      INTEGER REPMAC*2/12/,REPRC*2,REPTYP*4,REPCB*4,REPQUE*2,
     1RPVAL*2,RPECB*4,RPRES*4,RPTCB*4,RPPRM*4
      COMMON/WAIT/WTMAC,WTRC,WTECB
      INTEGER WTMAC*2/60/,WTRC*2/0/,WTECB/0/
      END
```

(The above common areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
      LOGICAL QPOS*1/Z04
```

Example 1

This example shows the method for purging a REPATCH control block, should a work request fail to be executed. The example begins with the PATCH-WAIT which is notified that a REPATCH is needed.

```
      CALL DPPPIF(WTMAC)
      IF(WTRC.NE.68) GO TO 100
      REPCB=WTECB
      REPTYP=8
      CALL DPPPIF(REPMAC)
 200  CONTINUE
```

Example 2

This example demonstrates the method of altering a REPATCH control block. In this case, the REPATCH will place the work request in the front of the work queue. As in Example 1, this example begins with a PATCH-WAIT.

```
 100  CALL DPPPIF(WTMAC)
      IF(WTRC.NE.68)GO TO 200
      RPCB=WTECB
      RPTYP=0
      CALL DPPPIF(RPMAC)
      CALL ORBIT(RPFLG,QPOS)
      WTECB=0
      RPTYP=4
      CALL DPPPIF(RPMAC)
      IF(RPRC.LT.8) GO TO 100
 200  CONTINUE
```

FORTRAN-PRIME Interface

The PTIME service provides two different functions, return current time and PATCHes, issued on a time-queue basis. The following FORTRAN statements define the two different parameter lists for this service:

```
C
C COMMON NAMED 'PTIMR'--PARAMETER TABLE FOR PTIME
 COMMON/PTIMR/PTRMAC
    INTEGER*2 PTRMAC
 COMMON/PTIMR/PTRC
    INTEGER*2 PTRC
 COMMON/PTIMR/PTRTYP
    INTEGER*4 PTRTYP
 COMMON/PTIMR/PTRTIM
    INTEGER*4 PTRTIM
 COMMON/PTIMR/PTRARY
    INTEGER*4 PTRARY
C END OF COMMON NAMED 'PTIMR'
```

PTRMAC
A halfword binary value of 4 to identify to the interface routine the requested service.

PTRC
A halfword binary value set to zero by the interface routine.

PTRTYP
A halfword binary value identifying the PTIMR service being requested. For this parameter list it is always zero.

PTRTIM
A fullword binary field which will contain the current time of day in 10-millisecond units when the interface routine returns.

PTRARY
A fullword field which will contain the address of the time array DPPCTIMA when the interface routine returns.

```
C
C COMMON NAMED 'PTIME--PARAMETER TABLE FOR PTIME
 COMMON/PTIME/PTIMAC
    INTEGER*2 PTIMAC
 COMMON/PTIME/PTIRC
    INTEGER*2 PTIRC
 COMMON/PTIME/PTITYP
    INTEGER*4 PTITYP
 COMMON/PTIME/PTISTR
    INTEGER*4 PTISTR
 COMMON/PTIME/PTITVL
    INTEGER*4 PTITVL
 COMMON/PTIME/PTIEND
    INTEGER*4 PTIEND
 COMMON/PTIME/PTIPAT
    INTEGER*4 PTIPAT
 COMMON/PTIME/PTIPRM
    INTEGER*4 PTIPRM
 COMMON/PTIME/PTIS
    LOGICAL*1 PTIS
 COMMON/PTIME/PTIP
    LOGICAL*1 PTIP
 COMMON/PTIME/PTIE
    LOGICAL*1 PTIE
C END OF COMMON NAMED 'PTIME'
C
```

PTIMAC
A halfword binary value of 4 to identify to the interface routine the requested service.

PTIRC
A halfword binary field containing a binary number return code from the service.

PTITYP
A fullword binary value identifying the requested PTIME service. Values may be 4, 8, or 12. If 4, a PTIME queue element (PTQE) is created which controls the PATCHes issued according to the PTIME requested. Since the PTQE exists independently of the creating task and may be modified (8) or deleted (12), the PTQE is referred to by task name, entry point name, and the PATCH ID number in the problem parameter list. Either task name or entry point name must be given for a modify (8) or delete (12). However, if only a task or entry point name is specified, all PTQEs with that name are deleted or modified.

PTISTR
A fullword binary value specifying the time in 10-millisecond units of the first PATCH. The flag bit in PTISTR defines the content of this field, according to the following table:

X'04' -- The first PATCH will be issued at current time plus the value of PTISTR.

X'02' -- The first PATCH will be issued when current time equals the value in PTISTR. If PTISTR is less than current time, the first PATCH will occur the next day.

X'01' -- The time of the first PATCH is calculated by assuming PTISTR contains the time of day, except that the value in PTITVL is added to PTISTR until that value is greater than current time.

PTITVL*
A fullword binary value specifying the interval in 10-millisecond units between successive PATCHes.

PTIEND**
A fullword binary value specifying when the PTQE is to be deleted. The flag bit in PTIE defines the content of this field.

X'08' -- PTIEND contains the count of the number of PATCHes to be issued by this PTQE.

X'04'** -- PTIEND contains a time value in 10-millisecond units, when added to current time equals the stop time.

X'02'** -- PTIEND contains the stop time in 10-millisecond units.

X'01'** -- The stop time is calculated by assuming PTIEND contains the time of day in 10-millisecond units except that the value in PTITVL is added to PTIEND until the value is greater than current time.

*All time units are in 10-millisecond units and must not exceed 24 hours.

**Regardless of what value is calculated start time (see PTISTR above), a 24-hour value is added to the stop time until the stop time exceeds the start time.

Note: If PTIEND and PTIE are zero, the PTIME is assumed infinite, and PATCHes will be issued until the PTQE is modified or deleted.

PTIPAT

Contains the address of the supervisor portion of the PATCH parameters. The options provided will be used by PTIME to issue PATCHes based on the above time options. If the common area PATCH is used as defined in the FORTRAN PATCH Interface write-up, the parameter must point to PATASK(1). All information desired for the PATCH by PTIME must be supplied prior to CALLing the interface routine. RESTRICTION: Queue position of DPATCH (X'02' in PATFLG) is not permitted.

PTIPRM

Contains the address of the parameter list passed by PTIME's PATCH. See FORTRAN PATCH write-up for formats.

Note: If this parameter list is greater than 8 bytes, the interface routine will move it to a GETMAIN area to be FREEMAINed when the PTQE is destroyed.

PTIS

A 1-byte logical field containing the flag which defines the content of PTISTR. See PTISTR above for flag definitions.

PTIP

A 1-byte logical field containing the flag which controls the kind of DPATCH which will be issued when the PTQE is destroyed. Flags at a PTIME delete (12) will override the flags when the PTQE was created (4) or last modified (8). Only one flag may be set.

X'08' -- Task is deleted regardless of its condition.

X'04' -- Task is deleted when its work queue becomes empty.

X'02' -- Task is deleted only if its work queue is empty.

X'01' -- Task is deleted immediately or when the current work queue, if executing, completes. Any work queue to the task is posted as deleted.

PTIE

A 1-byte logical field containing the flag which defines the content of PTIEND or zero. See PTIEND above for flag definitions.

The PTIME facilities are invoked by CALLing DPPPIF with the properly completed parameter list. Examples 1 through 4 assumed the following FORTRAN statements about COMMON area, variables, and constants:

```
 BLOCK DATA
 COMMON/PATCH/PATMAC,PATRC,PATPRM,PATASK(2),
1PATEP(2),PATNAM(2),PATQ,PATV,PATECB,PATRES(2),
1PATCBX,PATFLG
 INTEGER PATMAC*2/0/,PATRC*2/0/,PATPRM,PATQ*2/1/,
1PATV*2/0/,PATECB,PATRES,PATCBX
 LOGICAL PATASK*4/'     ',1'     '/,PATEP*4
1PATNAM*4/'     ','     '/,PATFLG*1
 COMMON/PTIME/PTIMAC,PATIRC,PTITYP,PTISTR,PTITVL,
1PTIEND,PTIPAT,PTIPRM,PTIS,PTIP,PTIE
 INTEGER PTIMAC*2/4/,PTIRC*2/0/,PTITYP,PTISTR,PTITVL,
1PTIEND,PTIPAT,PTIPRM,
 LOGICAL PTIS*1,PTIP*1,PTIE*1
 COMMON/PTIMR/PTRMAC,PTRC,PTRTYP,PTRTIM,PTRARY
 INTEGER PTRMAC*2/4/,PTRC*2/0/,PTRTYP/0/,PTRTIM,PTRARY
 END
```

(The above common areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
      LOGICAL TOD*1,/Z02/,TN*4(2)/'TIME','TEST'/,
     1TT*4(2)/'TTES','T    '/
      LOGICAL QPOS*1/Z04/,REL*1/Z01/,CNT*1/Z08/,ADJ*1/Z04/
      LOGICAL PU*1/Z01/,DEL*1/Z01/
      COMMON/PROBL/PRBLNG,ID,PROBP1
      INTEGER PRBLNG*2,ID*2,PROBP1*4
```

Include the preceding common areas in MAIN area also.

Example 1

In Example 1, the program uses the COMMON PTIMR to obtain the current time. The current time is used to set the start time in PTISTR for PATCHes by PTIME, at current time plus 1 hour. The interval is set to 1 hour, and the last PATCH is to occur 3 hours later. The PATCH parameters are set to create the task TIMETEST with a work queue length of 5, and a dispatching priority of 15 less than the PTIME task. The PATCH will execute program TTEST and delete it when the processing of each work request completes. The parameters are passed with a PATCH ID of 10.

```
      CALL DPPPIF(PTRMAC)
      PTIPAT=IADDR(PATASK(1))
      PTIPRM=IADDR(PRBLNG)
      PTISTR=PTRTIM + 360000
      CALL ORBIT(PTIS,TOD)
      PTITVL=360000
      PTIEND=PTISTR + 1080000
      CALL ORBIT(PTIE,TOD)
      DO 100 I=1,2
      PATASK(I)=TN(I)
  100 PATEP(I)=TT(I)
      PATQ=5
      PATV=15
      CALL ORBIT(PATFLG,DEL)
      PRBLNG=4
      ID=10
      PTITYP=4
      CALL DPPPIF(PTIMAC)
```

Example 2

In example 2, the PTQE built by Example 1 will be modified (TYPE=8) to start the PATCHes 15 seconds after this PTIME is issued, the interval is changed to once a minute, and the stop time is changed to never end. The program will not be deleted when a work request is finished processing and the work request will be queued first. The PATCH ID will be changed to 5. Note that all parameters must be specified, as a modify acts as a replace. All COMMONs are initially as defined.

```
      PTITYP=8
      PTIPAT=IADDR(PATASK(1))
      PTIPRM=IADDR(PRBLNG)
      PTISTR=1500
      CALL ORBIT(PTIS,REL)
      PTITVL=6000
      DO 100 I=1,2
      PATASK(I)=TN(I)
 100  PATEP(I)
      PATQ=5
      PATV=15
      CALL ORBIT(PATFLG,QPOS)
      PRBLNG=4
      ID=5
      CALL DPPPIF(PTIMAC)
```

Example 3

Example 3 shows the use of the adjusted time facility of PTIME. The first PATCH is to occur at 5 A.M., or within 30 minutes after the PTIME was issued and at 30-minute intervals for six times. The task is to be deleted immediately when the PTQE is destroyed.

```
 CALL ORBIT(PTIP,PU)
 PTISTR=180000
 CALL ORBIT(PTIS,ADJ)
 PTITVL=180000
 PTIEND=6
 CALL ORBIT(PTIE,CNT)
       .
       .
       .
 PATCH parameters
       .
       .
       .
 PROBLEM parameters
       .
       .
       .
 PTITYP=4
 CALL DPPPIF(PTIMAC)
```

Example 4

Example 4 shows the method for deleting a PTQE. Since the function of this PTIME service request is to locate the PTQE to be destroyed, only the parameters required to identify the PTQE need be given. In this case, the task is to be DPATCHed as well.

```
      PTITYP=12
      CALL ORBIT(PTIP,PU)
      PTIPAT=IADDR(PATASK(1))
      PTIPRM=IADDR(PRBLNG)
      DO 100I=1,2
      PATASK(I)=TN(1)
 100  PATEP(I)=TT(I)
      ID=10
      CALL DPPPIF(PTIMAC)
```

FORTRAN-MESSAGE Interface

The MESSAGE service is used to cause a predefined message to be printed or displayed. The message must have been defined through the offline utility system using the DEFMSG macro.

The following FORTRAN statements define the parameter list for this service:

```
C
C COMMON NAMED 'MESSAG'--PARAMETER TABLE FOR MESSAGE
 COMMON/MESSAG/MESMAC
    INTEGER*2 MESMAC
 COMMON/MESSAG/MESRC
    INTEGER*2 MESRC
 COMMON/MESSAG/MESNUM
    INTEGER*2 MESNUM
 COMMON/MESSAG/MESACT
    LOGICAL*1 MESACT
 COMMON/MESSAG/MESWT
    LOGICAL*1 MESWT
 COMMON/MESAG/MESRES
    INTEGER*4 MESRES
 COMMON/MESSAG/MESDAT
    INTEGER*4 MESDAT
 COMMON/MESSAG/MESRTE
    INTEGER*2 MESRTE(8)
 COMMON/MESSAG/MESVAR
    INTEGER*4 MESVAR(10)
C END OF COMMON NAMED 'MESSAG'
C
```

MESMAC
A halfword binary value of 40 identifying to the interface routine the requested service.

MESRC
A halfword field containing a binary number return code from the MESSAGE service routine. See MESSAGE macro write-up for valid return codes.

MESNUM
A halfword binary value from 1 to 999 identifying the message requested.

MESACT
A 1-byte logical field to be appended to the message number. I denotes information, A denotes action is required, and D denotes that a decision is required. Zero will indicate that the message definition default should be used.

MESWT
A 1-byte flag field using a X'08' to indicate the program's decision to WAIT for the message to be sent. A X'00' implies NO WAIT.

MESRES
A fullword binary field reserved for the interface routine.

MESDAT
A fullword binary field containing the address of an area where the service routine will place the formatted message for use by the program.

MESRTE
An array of eight halfword binary numbers representing the devices on which the message will appear or will be printed. All unused entries must be zero or 255. Values must range from 1 to 254. Entries with a zero will use the message definition default routing code.

MESVAR
An array of 10 fullwords containing addresses of message variables to be inserted into the message. All unused entries must be zero. Only consecutive non-zero entries will be used.

The following example requests the MESSAGE service to output to route code (1) message number 37 with a variable text field of "END OF TEST." The message number will have an action code of "I" appended to identify the message as an advisory. The program will wait for the message to be transmitted.

```
      BLOCK DATA
C     FORTRAN MESSAGE EXAMPLE
      COMMON/MESSAG/MESMAC,MESRC,MESNUM,MESACT,MESWT,MESRES,MESDAT,
     1MESRTE(8),MESVAR(10)
      INTEGER MESMAC*2/40/,MESRC*2/0/,MESNUM*2,MESRES,
     1MESDAT,MESRTE,MESVAR
      LOGICAL MESACT*1/100/,MESWT*1
      END
```

(The above COMMON areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
      LOGICAL A*4(4)/'bEND', 'bOFb','TEST',bbb'/,ACT*1/'I'/,
      WT*1/Z80/
      MESNUM=37
      MESACT=ACT
      CALL ORBIT(MESWT,WT)
      MESRTE(1)=1
      MESVAR(1)=IADDR(A(1))
      MESVAR(2)=0
      CALL DPPPIF(MESMAC)
```

FORTRAN-RECORD Interface

The RECORD facility provides a method for writing data to a sequential data set. The data can be retrieved at a later time for offline processing (see section on data playback). The following FORTRAN statements define the parameter list for this service:

```
C
C COMMON NAMED 'RECORD'--PARAMETER TABLE FOR RECORD
 COMMON/RECORD/RECMAC
    INTEGER*2 RECMAC
 COMMON/RECORD/RECRC
    INTEGER*2 RECRC
 COMMON/RECORD/RECCNT
    INTEGER*4 RECCNT
 COMMON/RECORD/RECDAT
    INTEGER*4 RECDAT
 COMMON/RECORD/RECID
    INTEGER*2 RECID
C END OF COMMON NAMED'RECORD'
C
```

RECMAC
A halfword binary value of 56 identifying to the interface routine the requested service.

RECRC
A halfword field containing a binary number return code from the RECORD service routine. See RECORD macro write-up for valid return codes.

RECCNT
A fullword binary field containing the number of data bytes to be recorded. A maximum value of 65535 may be specified.

RECDAT
The address of the data to be recorded.

RECID
A halfword binary number from 1 to 4095 which identifies the data being recorded.

The following example uses RECORD to write the entire 100 fullwords from array ANNUAL with an ID OF 100.

```
C  FORTRAN RECORD EXAMPLE
      BLOCK DATA
      COMMON/RECORD/RECMAC,RECRC,RECCNT,RECDAT,RECID
      INTEGER RECMAC*2/56/,RECRC*2/0/,RECCNT,RECDAT,RECID*2/0/
      END

      (The above COMMON areas should be repeated in the main program
      without data initialization. The following statements are
      in MAIN only.)

      INTEGER ANNUAL(100)
              .
              .
              .
      RECCNT=400
      RECDAT=IADDR(ANNUAL(1))
      RECID=100
      CALL DPPPIF(RECMAC)
```

FORTRAN-GETARRAY/PUTARRAY Interface

This FORTRAN interface provides the programmer the facilities of the GETARRAY and PUTARRAY services. The following FORTRAN statements define the interface parameter list:

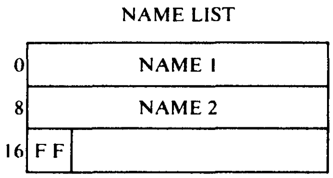
```
C
C COMMON NAMED'ARRAY'--PARAMETER TABLE FOR GETARRAY AND PUTARRAY
 COMMON/ARRAY/ARMAC
    INTEGER*2 ARMAC
 COMMON/ARRAY/ARRC
    INTEGER*2 ARRC
 COMMON/ARRAY/ARNAM
    INTEGER*4 ARNAM
 COMMON/ARRAY/ARAREA
    INTEGER*4 ARAREA
 COMMON/ARRAY/ARNADD
    INTEGER*2 ARNADD
 COMMON/ARRAY/ARADD
    INTEGER*2 ARADD
 COMMON/ARRAY/ARTYPE
    INTEGER*2 ARTYPE
C END OF COMMON NAMED 'ARRAY'
C
```

ARMAC
A halfword binary value of 16 identifying the service required to the interface routine.

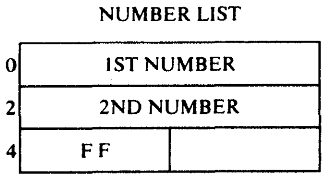
ARRC
A halfword binary field containing the return code from the array service routine. See GETARRAY and PUTARRAY macro write-ups for possible values.

ARNAM
A fullword field containing the address of one of the following based on the specifications implied by the value of ARTYPE.

a. If ARTYPE specifies the 'NAME LIST' option for ARNAM (see Figure 2-21), then ARNAM contains the address of a list of 8-character array names followed by an X'FF' after the last name where the next name would start. ARNADD contains the value to be added to the list address to locate the next array name.



b. If ARTYPE specifies 'NUMBER LIST' then ARNAM contains the address of halfword binary array numbers followed by a X'FF' after the last array number where the next number would start. ARNADD contains the value to be added to the list address to locate the next array number in the list.



c. If ARTYPE specifies 'ADDRESS LIST', then ARNAM contains the address of a list of array addresses as returned from a previous GETARRAY execution. The list must be terminated by a fullword binary value of -1 after the last array address where the next address would be located. ARNADD contains the value to be added to the list address to locate the next array address.

ADDRESS LIST

			4	6	8	10
0	FLAG	A(1ST ARRAY)	NO. BLKS	SIZE	NO. ITEMS	
+ARADD	FLAG	A(2ND ARRAY)	NO. BLKS	SIZE		
	FFFFFFFF					

This list is the same as returned as the find list specified below with the addition of the termination flag which must be added by the user.

ARAREA
A fullword field containing the address of one of the following based on the specifications implied by the value of ARTYPE.

a. If ARTYPE specifies the "DATA LIST' option for ARAREA (see Figure 2-21), then ARAREA contains the address of a list of addresses into or from which the data of the specified arrays (see ARNAM above) is to be moved. ARADD contains the value to be added to the list address to locate the next data area address in the list.

DATA AREA ADDRESS LIST

0	A(1ST DATA AREA)
ARADD*1	A(2ND DATA AREA)
ARADD*2	A(3RD DATA AREA)

b. If ARTYPE specifies 'FIND LIST', then ARAREA contains the address of a list of 10-byte fields to be filled: a flag byte (see GETARRAY macro write-up), a 3-byte array address, a halfword block count, a halfword array size or block size, and a halfword item count. ARADD contains the value to be added to the list address to locate the next entry in the list. The minimum value for ARADD under this option is 8, in which case, the item count halfword will not be in the list.

FIND LIST

		1	4	6	8
0	FLG	ARRAY ADDR	NO. BLKS	SIZE	NO. ITEMS
ARADD*1	FLG	ARRAY ADDR	NO. BLKS	SIZE	NO. ITEMS
ARADD*2	FLG	ARRAY ADDR	NO. BLKS	SIZE	NO. ITEMS

c. If ARTYPE of addresses specifies 'SPEC LIST', the ARAREA contains the address of a list of areas to be filled in by the service routine. Each area will receive a 16-byte field for each item in the array. These 16-byte fields will contain an 8-byte item name, a 1-byte item length, a 1-byte data type, a halfword array displacement to the start of the item, a halfword array ID, and a halfword number identifying the number of identical and sequential items defined by this entry. ARADD contains the value to be added to the list address to locate the next 16-byte field.

ARRAY SPECIFICATIONS LIST

		8	9	10	12	14
0	ITEM NAME	LNG	TYPE	DISP.	AID	REPT
16	ITEM NAME	LNG	TYPE	DISP	AID	REPT
32	ITEM NAME	LNG	TYPE	DISP	AID	REPT

ARNADD
A halfword value added to ARNAME to locate the next entry in the list. A value must be specified.

ARADD
A halfword value added to ARAREA to locate the next entry in the list. A value must be specified.

ARTYPE
A halfword binary value specifying the array service options selected. The values (given in the tables below) identify the contents of ARNAME and ARAREA, either a GETARRAY or PUTARRAY, the array (i.e., DATALIST, ADDRLIST, or SPECLIST), and the desired protection for GETARRAYs (PROTECT or RISK).

DATALIST
Specifies that the contents of the arrays are to be returned (GETARRAY) or updated (PUTARRAY).

ADDRLIST
Specifies that a 'FIND LIST' entry is to be completed for each array name or number in the list. This option is valid for virtual storage resident arrays only.

SPECLIST
Specifies that a 'SPEC LIST' entry is to be completed for each item of each array name or number in the list.

PROTECT
Specifies that the array service will be locked during processing to prevent changes from altering results.

RISK
Specifies that the array service will be processed regardless of the possibility of parallel processing changing array content.

ARNAM	ARAREA	SERVICE REQUESTED	PROTECTION REQUESTED	ARTYPE VALUE
A(NAME LIST)	A(DATA LIST)	DATA LIST	PROTECT	16
A(NAME LIST)	A(DATA LIST)	DATA LIST	RISK	17
A(NAME LIST)	A(SPEC LIST)	SPEC LIST	PROTECT	20
A(NAME LIST)	A(SPEC LIST)	SPEC LIST	RISK	21
A(NAME LIST)	A(FIND LIST)	ADDR LIST	PROTECT	34
A(NAME LIST)	A(FIND LIST)	ADDR LIST	RISK	35
A(ADDR LIST)	A(DATA LIST)	DATA LIST	PROTECT	48
A(ADDR LIST)	A(DATA LIST)	DATA LIST	RISK	49
A(NUMBER LIST)	A(DATA LIST)	DATA LIST	PROTECT	80
A(NUMBER LIST)	A(DATA LIST)	DATA LIST	RISK	81
A(NUMBER LIST)	A(SPEC LIST)	SPEC LIST	PROTECT	84
A(NUMBER LIST)	A(SPEC LIST)	SPEC LIST	RISK	85
A(NUMBER LIST)	A(FIND LIST)	ADDR LIST	PROTECT	98
A(NUMBER LIST)	A(FIND LIST)	ADDR LIST	RISK	99

Figure 2-21. GETARRAY Service

A(NAME LIST)	A(DATA LIST)	DATA LIST	N/A	128
A(ADDR LIST)	A(DATA LIST)	DATA LIST	N/A	144
A(NUMBER LIST	A(DATA LIST)	DATA LIST	N/A	176

Figure 2-22. PUTARRAY Services

The GETARRAY/PUTARRAY services are invoked by CALLing DPPPIF with the properly completed parameter list.

The following example shows the use of the array services in locating the array B, reading in the item specifications, reading the entry array into FORTRAN core, and updating the array.

```
      BLOCK DATA
C     FORTRAN GET/PUT-ARRAY EXAMPLE
C
      COMMON/ARRAY/ARMAC,ARRC,ARNAM,ARAREA,ARNADD,ARADD,ARTYPE
      INTEGER ARMAC*2/0/,ARRC*2/0/,ARNAM,ARAREA,ARNADD*2/8/,
     1ARADD*2/4/,ARTYPE*2/16/
      END
```

(The above COMMON areas should be repeated in the main program without data initializtion. The following statements are in MAIN only.)

```
      COMMON/ARAY/ANAME(2),AEND,AFIND(6,2),ACORE
      INTEGER END*4,AFIND*2,ACORE*4
      LOGICAL ANAME*4
      COMMON/AITM/INAME(16,255)
      LOGICAL INAME*1
      EQUIVALENCE (AFIND(1,1),ADRA(1,1)
      INTEGER ADRA(3,2)
      EQUIVALENCE (INAME(1,1),SPEC(1,1))
      INTEGER SPEC*2(8,255)
      COMMON/AREA/DATA(16,255)
      LOGICAL DATA*1
      LOGICAL A*4(2)/'B    ','    '/
      ANAME(1)=A(1)
      ANAME(2)=A(2)
      AEND=1
      ARNAM=IADDR(ANAME(1))
      ARNADD=8
      ARAREA=IADDR(AFIND(1,1))
      ARADD=12
      ARTYPE=35
C     BUILD FIND LIST
      CALL DPPPIF(ARMAC)
      ACORE=IADDR(INAME(1,1))
      ARAREA=IADDR(ACORE)
      ARADD=4
      ARTYPE=21
C     BUILD ITEM SPEC LIST
      CALL DPPPIF(ARMAC)
      ACORE=IADDR(DATA(1,1))
      ARTYPE=16
C     READ ARRAY
      CALL DPPPIF(ARMAC)
      ARTYPE=128
C     UPDATE ARRAY
      CALL DPPPIF(ARMAC)
```

FORTRAN-GETITEM/PUTITEM Interface

This FORTRAN interface provides the programmer the facilities of the GETITEM and PUTITEM services. The following FORTRAN statements define the interface paramater list:

```
C
C COMMON NAMED 'ITEM'--PARAMETER TABLE FOR GETITEM AND PUTITEM
 COMMON/ITEM/ITMAC
    INTEGER*2 ITMAC
 COMMON/ITEM/ITMRC
    INTEGER*2 ITMRC
 COMMON/ITEM/ITMNAM
    INTEGER*4 ITMNAM
 COMMON/ITEM/ITMDAT
    INTEGER*4 ITMDAT
 COMMON/ITEM/ITMNAD
    INTEGER*2 ITMNAD
 COMMON/ITEM/ITMDAD
    INTEGER*2 ITMDAD
 COMMON/ITEM/ITMTYP
    INTEGER*2 ITMTYP
C END OF COMMON NAMED 'ITEM'
C
```

ITMAC
A halfword binary value of 20 identifying the service required to the interface routine.

ITMRC
A halfword field containing a binary number return code from the item service routine. See GETITEM and PUTITEM macro write-ups for possible values.

ITMNAM
A fullword field containing the address of one of the following based on the specifications implied by the value of ITMTYP.

a. If ITMTYP specifies 'NAMELIST', the ITMNAM contains the address of a list of 8-character item names followed by a X'FF' after the last name where the next name would start.

   ITMNAD contains the value to be added to the list address to locate the next item name.

NAME LIST

0	NAME 1	
+ITMNAD	NAME 2	
+ITMNAD*2	F F	

b. If ITMTYP specifies 'ADDRESS LIST', the ITMNAM contains the address of a list of item addresses as returned from a previous execution. The list must be terminated by a fullword of -1 where the next address would be in the list. ITMNAD contains the value to be added to the list address to locate the next item address in the list.

ADDRESS LIST

0	LENGTH	A(ITEMA)
+ITMNAD	LENGTH	A(ITEMB)
+ITMNAD*2	FFFFFFFF	

ITMDAT
A fullword field containing the address of one of the following based on the specifications implied by the value of ITMTYP.

a. If ITMTYP specifies 'DATA LIST', the ITMDAT contains the address of a data area into or from which data is moved. ITMDAD contains the value to be added to the data area addresss to locate the area for the next item. If ITMDAD is zero, the item length is used to locate the next item data area.

b. If ITMTYP specifies 'ADDR LIST', the ITMDAT contains the address of a list of 4-byte entries into which an item length and address is stored for each item specified in the 'NAME LIST.' The list must contain room for one more entry to allow the service routine to store an end of list X'FF.' ITMDAD contains the value to be added to the list address of locate the next entry.

ADDRESS LIST

0	LENGTH	ITEM ADDRESS
4	LENGTH	ITEM ADDRESS
8	FF	FF FF FF

c. If ITMTYP specifies 'SPECLIST', the ITMDAT contains the address of a list of 4-byte entries each containing the item length, flags identifying data type, and an array displacement to the first byte of the item. ITMDAD contains the value to be added to the list address of locate the next entry.

Item Specification List

0	8	9	10	12	14	16
ITEM NAME	LNG	TYPE	DISP.	AID	REPT	
ITEM NAME	LNG	TYPE	DISP.	AID	REPT	
ITEM NAME	LNG	TYPE	DISP.	AID	REPT	

ITMNAD
A halfword binary value added to the list address in ITMNAM to locate the next entry   A value must be A value must be specified.

ITMDAD
A halfword binary value added to the list address in ITMDAT to locate the next entry. A value must be specified unless ITMTYP specifies 'DATA LIST', in which case zero may be used.

ITMTYP
A halfword binary number specifying the item service options selected. The values (given in the figures 2-23 and 2-24) identify the kind of service (i.e., DATALIST,ADDRLIST, or SPECLIST), if it is a GETITEM or PUTITEM, and if GETITEM is protected (PROTECT or RISK). A value must be specified.

DATALIST
Specifies the content of the item is to be moved to or updated from the data area.

ADDRLIST
Specifies the item 'ADDRESS LIST' is to be built for each named item.

SPECLIST
Specifies the item 'SPECIFICATION LIST' is to be built for each named item.

PROTECT
Specifies the GETITEM service will ensure data integrity during processing.

RISK
Specifies the GETITEM service will process the request regardless of the possibility of parallel processing updating the content of the named item(s).

Note: DATALIST and ADDRILIST are invalid service requests for direct access resident arrays.

ITMNAM	ITMDAT	SERVICE REQUESTED	PROTECTION REQUIRED	ITMTYP VALUE
A(NAME LIST)	A(DATA LIST)	DATA LIST	PROTECT	136
A(NAME LIST)	A(DATA LIST)	DATA LIST	RISK	137
A(NAME LIST)	A(ADDR LIST)	ADDR LIST	PROTECT	138
A(NAME LIST)	A(ADDR LIST)	ADDR LIST	RISK	139
A(NAME LIST)	A(SPEC LIST)	SPEC LIST	PROTECT	140
A(NAME LIST)	A(SPEC LIST)	SPEC LIST	RISK	141
A(ADDR LIST)	A(DATA LIST)	DATA LIST	PROTECT	152
A(ADDR LIST)	A(DATA LIST)	DATA LIST	RISK	153

Figure 2-23. GETITEM Services

A(NAME LIST)	A(DATA LIST)	DATA LIST	N/A	184
A(ADDR LIST)	A(DATA LIST)	DATA LIST	N/A	200

Figure 2-24. PUTITEM Services

The GETITEM/PUTITEM services are invoked by CALLing DPPPIF with a properly completed parameter list.

The following example will use the item service to obtain the addresses, specifications, and data for a list of five items from the same array and update them in the array.

```
      BLOCK DATA
C     FORTRAN GET/PUT-ITEM EXAMPLE
      COMMON /ITEM/ ITMAC,ITMRC,ITMNAM,ITMDAT,ITMNAD,ITMDAD,
     1ITMTYP
      INTEGER ITMAC*2/20/,ITMRC*2/0/,ITMNAM,ITMDAT,ITMNAD*2,
     1ITMDAD*2,ITMTYP*2/0/
      COMMON /AREA/ DATA(16,5)
      LOGICAL*1 DATA
      COMMON /N/ NAME(2,5),END
      INTEGER END/-1/
      LOGICAL*4 NAME/1B01 ','    ','B03 ','    ','B05 ','    ',
     1'B07b','bbb','B09b','bbbb'/
      END
```

(The above common areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
      INTEGER ADR(6)
      LOGICAL*1 LNG(4,5)
      ITMNAM=LADDR(NAME(1,1))
      ITMNAD=8
      ITMDAT=IADDR(ADR(1))
      ITMDAD=4
      ITMTYP=139
C     FIND ARRAY ITEMS
      CALL DPPPIF(ITMAC)
      ITMDAT=IADDR(LNG(1,1))
      ITMTYP=141
C     GET ITEM SPECS
      CALL DPPPIF(ITMAC)
      ITMNAM=IADDR(ADR(1))
      ITMNAD=4
      ITMDAT=IADDR(DATA(1,1))
      ITMDAD=16
      ITMTYP=152
C     READ ITEMS BY ADDRESS
      CALL DPPPIF(ITMAC)
      ITMTYP=200
C     UPDATE ITEMS BY ADDRESS
      CALL DPPPIF(ITMAC)
```

FORTRAN-GETBLOCK/PUTBLOCK Interface

This FORTRAN interface provides the programmer the facilities of the GETBLOCK and PUTBLOCK services. The following FORTRAN statements define the interface parameter list:

```
C
C COMMON NAMED 'BLOCK'mPARAMETER TABLE FOR GETBLOCK AND
  PUTBLOCK

 COMMON/BLOCK/BLKMAC
    INTEGER*2 BLKMAC
 COMMON/BLOCK/BLKRC
    INTEGER*2 BLKRC
 COMMON/BLOCK/BLKNAM
    INTEGER*4 BLKNAM
 COMMON/BLOCK/BLKDAT
    INTEGER*4 BLKDAT
 COMMON/BLOCK/BLKADD
    INTEGER*2 BLKADD
 COMMON/BLOCK/BLKTYP
    INTEGER*2 BLKTYP

C END OF COMMON NAMED 'BLOCK'
C
```

BLKMAC
A halfword binary value of 24 identifying the service requested to the interface routine.

BLKRC
A halfword field containing a binary number return code from the blocked array service routine. Zero indicates successful completion while any non-zero indicates unsuccessful completion.

BLKNAM
A fullword field containing the address of one of the following based on the specifications implied by BLKTYP.

a. If BLKTYP specifies 'NAME LIST' the BLKNAM contains the address of a list of 8-character array names followed by a X'FF' in the first byte after the last name where the next name would start.

NAME LIST

0	NAME	
8	NAME	
16	FF	

b. If BLKTYP specifies 'NUMBER LIST', the BLKNAM contains the address of a list of halfword array number followed by a X'FF' in the first byte after the last number where the next number would start.

NUMBER LIST

0	NUMBER	
2	NUMBER	
4	FF	

BLKADD contains the value to be added to the list address to locate the next entry.

DATA AREA LIST

0	1	4
FLG	DATA AREA	BLK. NO.
FLG	DATA AREA	BLK. NO.
FLG	DATA AREA	BLK. NO.

FLG -- A 1-byte flag field. A X'40' indicates the last data area and block number for a specified array but not the end of the list. A X'80' indicates the last entry for the last array and the end of the list. A X'00' should appear in all other entries.

DATA AREA -- A 3-byte address of the area into or from which the specified array block is moved.

BLK. NO. -- A halfword binary number specifying the array block being moved.

BLKADD
A halfword binary value added to the contents of BLKDAT to locate the next entry in the list. If zero, a value of 6 is assumed.

BLKTYP
A halfword binary value specifying the blocked array service options selected. The value (given in the tables below) identify the contents of BLKNAM and if it is a GETBLOCK with or without protection (PROTECT or RISK) or a PUTBLOCK.

BLKNAM	BLKDAT	PROTECTION REQUESTED	BLKTYP VALUE
A(NAME LIST)	A(DATA LIST)	RISK	4
A(NUMBER LIST)	A(DATA LIST)	RISK	6
A(NAME LIST)	A(DATA LIST)	PROTECT	12
A(NUMBER LIST)	A(DATA LIST)	PROTECT	14

Figure 2-25. GETBLOCK Services

A(NAME LIST)	A(DATA LIST)	N/A	5
A(NUMBER LIST)	A(DATA LIST)	N/A	7

Figure 2-26. PUTBLOCK Services

The GETBLOCK/PUTBLOCK services as invoked by calling DPPPIF with a properly completed parameter list.

The following example will execute a GETBLOCK for block number 5 from array BLK1 and BLOKB. Then the blocks are written out to their respective arrays.

```
      BLOCK DATA
C     FORTRAN GET/PUT-BLOCK EXAMPLE
      COMMON /BLOCK/ BLKMAC,BLKRC,BLKNAM,BLKDAT,BLKADD,BLKTYP
      INTEGER BLKMAC*2/24/,BLKRC*2,BLKNAM,BLKDAT,BLKADD*2,
     1BLKTYP*2
      COMMON /N/ NAME(2,2),END
      LOGICAL*4 NAME/'BLK1','    ','BLOK','B   '/
      INTEGER*2 END/-1/
      END
```

(The above COMMON areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
      COMMON /LIST/ AREA(2,2)
      INTEGER*4 AREA
      EQUIVALENCE(AREA(1,1),NUM(1,1))
      INTEGER*2 NUM(4,2)
      COMMON /BLK/ DATA(256,2)
      LOGICAL*1 DATA
      LOGICAL*1 NEXT/Z40/,STOP/Z80/
      AREA (1,1)=IADDR(DATA(1,1))
      NUM(3,1)=5
      CALL ORBIT(AREA(1,1),NEXT)
      AREA(1,2)=IADDR(DATA(1,2))
      NUM(3,2)=5
      CALL ORBIT(AREA(1,1),STOP)
      BLKNAM=IADDR(NAME(1,1))
      BLKDAT=IADDR(AREA(1,1))
      BLKADD=8
      BLKTYP=12
C     READ BLOCK 5 OF ARRAYS BLK1 and BLOKB
      CALL DPPPIF(BLKMAC)
      BLKTYP=5
C     UPDATE BLOCK 5 IN ARRAYS
      CALL DPPPIF(BLKMAC)
```

FORTRAN-GETLOG Interface

This FORTRAN interface provides the programmer the facilities of the GETLOG service. The following FORTRAN statements define the interface parameter list.

```
C
C COMMON NAMED 'GETLOG' PARAMETER TABLE FOR GETLOG
C
 COMMON/GETLOG/GETMAC
    INTEGER*2 GETMAC
 COMMON/GETLOG/GETRC
    INTEGER*2 GETRC
 COMMON/GETLOG/GETYPE
    INTEGER*2 GETYPE
 COMMON/GETLOG/GETNO
    INTEGER GETNO*2
 COMMON/GETLOG/GETDAT
    INTEGER*4 GETDAT
 COMMON/GETLOG/GETCPY
    INTEGER*4 GETCPY
 COMMON/GETLOG/GETHD
    INTEGER*4 GETHD
 COMMON/GETLOG/GETNAM
    INTEGER*4 GETNAM
C END OF COMMON NAMED 'GETLOG'
C
```

GETMAC
A halfword value of 48 identifying the service required to the interface routine.

GETRC
A halfword binary field containing a binary number return code from the GETLOG service routine. See GETLOG macro write-up for possible values.

GETYPE
A halfword flags field indicating the requested options to the GETLOG service routine. Bits are numbered 0 to 7.

Bits 0, 1,
3, 5, and 77 -- Reserved.

Bit 2 -- See GETHD.

Bit 4 -- If on, the GETLOG service routine protects the data content across the service request.

Bit 6 -- If on, GETNO contains the array number of the log copy being read. If off, GETNAM contains the address of the array name.

Byte 2 -- Reserved.

GETNO
A halfword field containing the number of the array whose log copy is being read. Valid only if Bit 6 of GETYPE is on. If bit 6 is off, this field is zeroed by the interface routine.

GETDAT
A fullword field containing the address of the data area into which the log copy requested will be placed. The area must be large enough to hold the entire array and its 24-byte log header.

GETCPY
A fullword binary number used to determine which copy of a logged array, relative to the GETHD parameters, will be retrieved from the log data set.

GETHD
A fullword field containing one of the following based on Bit 2 in GETYPE

a. If Bit 2 is on, then GETHD contains the address of a 24-byte log header identifying the relative starting point to determine which copy of the array will be retrieved from the log data set.

b. If Bit 2 is off and GETHD is zero, then the current copy becomes the relative starting point.

c. If Bit 2 is off and GETHD is non-zero, then GETHD contains the address of a 6-byte time and day field. The first 4 bytes will contain a time in 10-millisecond units. The last two bytes contain a binary value from 1 to 366, representing the day of the year. This time and day will be used as a comparison value to establish a relative starting point to determine which copy of the array will be retrieved from the log data set.

GETNAM
A fullword address of the name of the named array, a log copy of which is being requested. Valid only if BIT 6 of GETYPE is off.

The GETLOG service is invoked by CALLing DPPPIF with a properly completed parameter list.

The following example will GETLOG the previous logged copy of array B referenced from the current copy.

```
      BLOCK DATA
C     FORTRAN GETLOG EXAMPLE
C
      COMMON/GETLOG/GETMAC,GETRC,GETYPE,GETNO,GETDAT,,GETCPY,
     1GETHD,GETNAM
      INTEGER GETMAC*2/48/,GETRC*2/0/,GETYPE*2/0/,GETNO*2,
     1GETDAT,GETCPY,GETHD,GETNAM
      END
```

(The above common areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
     COMMON/LOG/HEADR(12),LDATA(24)
      INTEGER*2 HEADR,LDATA
      LOGICAL A*4(2)/'B     ','       '/
          .
          .
          .
      GETDAT=IADDR(HEADR(1))
      GETNAM=IADDR(A(1))
      GETCPY=-1
      CALL DPPPIF(GETMAC)
```

FORTRAN-PUTLOG Interface

This FORTRAN interface provides the programmer the facilities of the PUTLOG service. The following FORTRAN statements define the interface parameter list.

```
C
C COMMON NAMED 'PUTLOG' PARAMETER TABLE FOR PUTLOG
C
 COMMON/PUTLOG/PUTMAC
    INTEGER*2 PUTMAC
 COMMON/PUTLOG/PUTRC
    INTEGER*2 PUTRC
 COMMON/PUTLOG/PUTNAM
    INTEGER*4 PUTNAM
 COMMON/PUTLOG/PUTHD
    INTEGER*4 PUTHD
 COMMON/PUTLOG/PUTYPE
    INTEGER*2 PUTYPE
 COMMON/PUTLOG/PUTBLK
    INTEGER*2 PUTBLK
C
C END OF COMMON NAMED 'PUTLOG'
C
```

PUTMAC
A halfword binary value of 44 identifying the requested service to the interface routine.

PUTRC
A halfword binary field containing a binary number return code from the PUTLOG service routine. See PPUTLOG macro write-up for possible values.

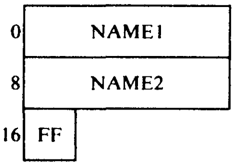
PUTNAM
A fullword containing the address of one of the following based on Bits 5 and 6 in PUTYPE.

a. If Bits 5 and 6 are zero (where bits in a byte are numbered 0 to 7), then PUTNAM contains the address of an 8-character array name.

b. If Bit 5 is off and Bit 6 is on, then PUTNAM contains the address of a halfword containing an array number.

c. If Bit 5 is on and Bit 6 is off, then PUTNAM contains the address of a list of 8-character array names. The first byte past the last valid entry must be set to X'FF' to indicate the end of the name list.

NAME LIST



d. If Bits 5 and 6 are on, then PUTNAM contains the address of a list of halfword binary array numbers. The first byte past the last valid entry must be set X'FF' to indicate the end of the number list.

NUMBER LIST

0	1ST NUMBER
2	2ND NUMBER
4	FF

PUTHD
A fullword field containing the address of one of the following based on Bits 2 and 3 in PUTYPE.

a. If Bits 2 and 3 are both off, then PUTHD must be zero.

b. If Bit 2 is on and Bit 3 is off, then PUTHD contains the address of an array logging header. Information in this logging header will identify the copy of the array which is to be replaced in the log data set. The logging header is a 24-byte control block which precedes the array, both as the array exists in virtual storage and as it is written to the logging array. The logging header which was retrieved as part of a previous GETLOG may be used to replace that copy in the log data set.

c. If Bit 2 is off and Bit 3 is on, the PUTHD contains the address of a user-constructed list of block numbers and storage addresses. The latest log copy will be modified. However, only the log block corresponding to the VS resident block specified will be updated.

0	1	4
FLG	DATA ADDRESS	BLK. NO.

FLG -- A 1-byte flag field.

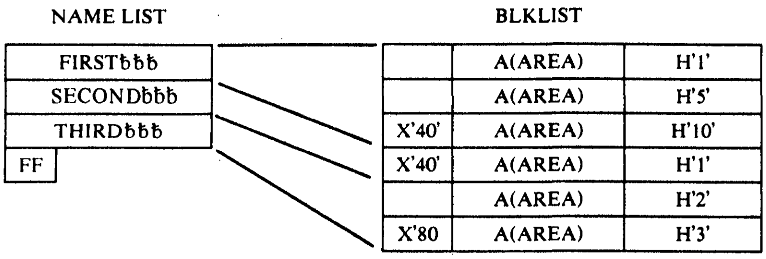
X'40' -- Indicates the last entry to be processed for a particular entry in the name or number list.

X'80' -- Indicates the last entry in the data list.

DATA ADDRESS -- Ignored.

BLK NO. -- The number assigned to the data block to be updated.

EXAMPLE: BLKLIST and Name List



NAME LIST

FIRSTbbb
SECONDbbb
THIRDbbb
FF

BLKLIST

	A(AREA)	H'1'
	A(AREA)	H'5'
X'40'	A(AREA)	H'10'
X'40'	A(AREA)	H'1'
	A(AREA)	H'2'
X'80	A(AREA)	H'3'

PUTYPE
A 2-byte flags field specifying the selected options.

Bits 0 and 1 -- Reserved.

Bits 2 and 3 -- See PUTHD.

Bit 4 -- If on, the PUTLOG is protected while processing.

Bits 5 and 6 -- See PUTNAM.

Bit 7 -- Must be on to indicate a PUTLOG.

Byte 2 -- Reserved.

PUTBLK
If flag bit 2 is off and Bit 3 is on, then the halfword value in this field is used to increment the address in PUTHD.

The PUTLOG service is invoked by CALLing DPPPIF with the properly completed parameter list.

The following example logs the array B as the current log copy.

```
       BLOCK DATA
C     FORTRAN PUTLOG EXAMPLE
C
      COMMON/PUTLOG/PUTMAC,PUTRC,PUTNAM,PUTHD,PUTYPE,
     1PUTBLK
      INTEGER PUTMAC*2/44/,PUTRC*2/0/,PUTNAM,PUTHD,PUTYPE*2/0/,
      1PUTBLK*2/0/
      END

      (The above COMMON areas should be repeated in the main program
      without data initialization.  The following statements
      are in MAIN only.)

      LOGICAL*4 A(2)/''B   ','      '/
      LOGICAL*1 PUT/Z01/
      CALL ORBIT(PUTYPE,PUT)
      PUTNAM=IADDR(A(1))
      CALL DPPPIF(PUTMAC)
```

FORTRAN-DUMPLOG Interface

This FORTRAN interface provides the programmer the facilities of the DUMPLOG service. The following FORTRAN statements define the interface parameter list:

```
C COMMON NAMED 'DUMPLG'--PARAMETER TABLE FOR DUMPLOG
 COMMON/DUMPLG/DPLMAC
    INTEGER*2 DPLMAC
 COMMON/DUMPLG/DPLRC
    INTEGER*2 DPLRC
 COMMON/DUMPLG/DPLTYP
    INTEGER*2 DPLTYP
 COMMON/DUMPLG/DPLNO
    INTEGER*2 DPLNO
 COMMON/DUMPLG/DPLSTR
    INTEGER*4 DPLSTR
 COMMON/DUMPLG/DPLEND
    INTEGER*4 DPLEND
 COMMON/DUMPLG/DPLDAT
    INTEGER*4 DPLDAT
 COMMON/DUMPLG/DPLDD
    LOGICAL*1 DPLDD(8)
 COMMON/DUMPLG/DPLIST
    INTEGER*4 DPLIST
C END OF COMMON NAMED 'DUMPLG'
C
```

DPLMAC
A halfword binary value of 52 identifying the requested service to the interface routine.

DPLRC
A halfword binary field containing a binary number return code from the DUMPLOG service routine. See DUMPLOG macro write-up for possible values.

DPLTYP
A halfword flags field indicating the requested options to the GETLOG service routine. Bits are numbered 0 to 7.

Bits 0, 1, 2, 4 and 7 -- Reserved

Bit 3 -- This flag specifies whether the dumped copies are to be written at the beginning of the dump data set (Bit 3 is on) or added to the existing dumped copies (Bit 3 is off). If the disposition parameter specified on the DD card statement for this data set is either OLD or SHR and the data set is empty, then the first DUMPLOG request must specify 'NEW' (Bit 3 is on). Specifying 'NEW' (Bit 3 is on) on subsequent DUMPLOG requests will position a direct access data set to record one and will cause a tape data set to force EOV before the log copies are written.

Bit 5 -- If on, specifies a list of array names or numbers is pointed to by DPLIST.

Bit 6 -- If on, specifies array number(s) is to be processed. If off, array name(s) is given for processing.

DPLNO
A halfword number which is the number of a numbered array to be dumped. Valid only if Bit 5 is off and Bit 6 of DPLTYP is on.

DPLSTR
A fullword which specifies the address of a 6-byte time and day field. The first four bytes will contain a time in 10-millisecond The last two bytes will contain a binary value from 1 to 266 representing the date of the year. The logged copies of the array will be searched until a copy is found with a log time equal to or greater than the start time specified. If this parameter is zero, dumping commences with the oldest logged copy of the array.

DPLEND
A fullword which specifies the address of a 6-byte time and day field formatted as in DPLSTR. The logged copies of the array will be dumped until the most recently logged copy has been dumped or until a copy is dumped whose log time is equal to or greater than the specified stop time. If this parameter is zero, dumping will terminate when the most recently logged copy of the array has been dumped.

Note: DUMPLOG will insert a byte of X'FF' into the first byte of the logging header of the last copy of each array dumped to the sequential data set to indicate the end of the dump of each array to the user delog routine.

DPLDAT
A fullword which specifies the address of a 256-byte user data area to be contained in the dump header for each array on the sequential dump data set.

DPLDD
Two 4-byte logical words containing the name of the data definition (DD) statement which describes a sequential data set to receive the dumped copies of the array(s) from the log data set. A name must be specified.

The output will consist of spanned variable length records. The blocksize of the data set defined by DPLDD must be at least 264 bytes but no more than 32,760 bytes. The blocksize should be large enough to contain one array copy, its log header, the user dump header, if any, and the variable length descriptor words (8 bytes) for maximum effiency.

DPLIST
A fullword containing the address of one of the following based on Bits 5 and 6 DPLTYP:

a. If Bits 5 and 6 are off, then DPLIST contains the address of an 8-character loggable array name to be dumped.

b. If Bit 5 is on and Bit 6 is off, then DPLIST contains the address of a list of loggable array names to be dumped.

Each name is eight characters long with a X'FF' after the last valid name as an end of list indicator.

NAME LIST

0	ARRAY NAME
8	ARRAY NAME
16	FF

c. If Bits 5 and 6 are on, then DPLIST contains the address of a list of halfword loggable array numbers. A X'FF' follows the last valid number as an end of list indicator.

NUMBER LIST

0	NUMBER
2	NUMBER
4	FF

The DUMPLOG service may be invoked by CALLing DPPPIF with a properly completed parameter list.

The following example will dump log array B at the beginning of the data set. All log copies of array B will be dumped starting with the oldest copy available.

```
C FORTRAN DUMPLOG EXAMPLE
  BLOCK DATA
  COMMON /DUMPLG/ DPLMAC,DPLRC,DPLTYP,DPLNO,DPLSTR,DPLEND,
 1DPLDAT,DPLDD(2),DPLIST
  INTEGER DPLMAC*2/52/,DPLRC*2/0/,DPLTYP*2/0/,DPLNO*2,
 1DPLSTR,DPLEND,DPLDAT,DPLIST
  LOGICAL DPLDD*4/'DUMP','LOG '/
  END
```

(The above common areas should be repeated in the main program without data initialization. The following statements are in MAIN only.)

```
  LOGICAL A*4(2)/'B    ','    '/,DISP*1/Z10/
      .
      .
      .
  CALL ORBIT(DPLTYP,DISP)
  DPLIST = IADDR(A(1))
  CALL DPPPIF(DPLMAC)
```

DUPLICATE DATA SET SUPPORT

The operation of the Special Real Time Operating System and associated subsystems is dependent upon several direct access data sets. Some of these, such as data base definitions, are only read in realtime execution, while others, such as history logs, are read and written. The Special Real Time Operating System provides the capability to use two identical copies of certain data sets to improve the total system availability. While this service is provided primarily for data sets which are used by the system, a limited capability is provided to the system user to utilize the duplicate data set support.

The principal purpose of the duplicate data set facility is to provide a backup copy of the data should the primary copy experience a failure. In maintaining this duplicate data set, the primary and backup are updated simultaneously during realtime processing. In case of failure of one copy, the system takes that copy out of service and uses the other copy. Appropriate messages are output to make the operator aware of the trouble.

Duplicate data set support (DDS) is a SYSGENable option which is selected by coding the DUPDISK macro at Special Real Time Operating System SYSGEN time. With DDS SYSGENed, a user can declare via JCL the data sets that are duplicates. The user programs include special I/O macro codes to use the DDS services. However, this does not prevent these programs from functioning when DDS is not supported, because the special macros default to their standard OS/VS1 counterparts for data sets not supported by DDS.

DDS services are in three logical areas:

1. Initialization

2. Pseudo-SVC routines

3. I/O CALL routines

Initialization analyzes the DDS input stream to determine which data sets are being declared as duplicate. A control table is established for properly declared duplicate data sets, and its address is placed in the Special Real Time Operating System SCVT.

The DDS pseudo-SVC routines are given control when the user requests an I/O function which is normally an SVC under standard OS access methods. Thus, the OPEN, CLOSE, BLDL, FIND, and STOW macro functions require corresponding DDS macros (OS macros preceded by DDS) which expand not to an SVC, but to a branch to the respective DDS routine. If the data set has been declared a duplicate, these routines will issue the SVC for both data sets; if not, these routines will issue the SVC only once. The DCBOFLGS and SVC return codes are provided to the user in either case.

DDS I/O call routines will be entered for all I/O requests (READ, WRITE, NOTE, POINT, and CHECK) to a data set that was opened with the DDS OPEN macro and was declared a duplicate. These routines will treat the request in the following manner: all requests to alter the data set are issued to both data sets, and all requests to read data are issued only to the primary data set. In the update mode, the read request is issued twice. In case of an incorrectable I/O failure, the failing data set is closed, and processing continues with the remaining data set. For double I/O failures, the user's SYNAD is given control. To prevent double I/O failures, the data sets should be on devices that are on different I/O channels.

The user declares which data sets are duplicates via JCL, by including the DD card DDSCTLIN. Each duplicate data set should be described by a separate 'DDSNAMES' card in the DDSCTIIN stream. The format of the DDSNAMES card is as follows:

```
 [DDS-DDNAME]     DDSNAMES     (DDNAME1,DDNAME2=OUT)
```

DDS-DDNAME
Is optional and must begin in column 1. It should be the DDNAME that will be referenced in all I/O macros for this duplicate data set. If left blank, its value will default to that supplied in the DDNAME1 field.

DDSNAMES
Is the required op code and should be preceded by at least 1 blank.

DDNAME1
Is the DDNAME of the DD card for the primary data set of the duplicate pair. This field is required.

DDNAME2
Is the DDNAME of the DD card for the backup data set of the duplicate pair. This field is required and the backup can be initialized out of service.

Certain DDS functions can be requested dynamically during realtime operation. These functions allow the user, through the input message processor, to:

- Create a backup
- Take a backup out of service
- Switch the primary and backup
- Replace the primary
- Compare primary and backup
- Give the status of the duplicate data sets.

The format of the replies required to invoke these routines is documented in the section entitled "DDSCNTRL Command. The input message command is DDSCNTRL.

The user can have his current primary data set copied to his backup to bring the backup to the same level as the primary. This operation requires that the backup data set be out of service for the copy operation. The user also may use the DDSCNTRL reply to take the backup out of service.

The DDSCNTRL reply may be used to cause the backup to become the primary and have the primary switched to an out of service backup. But backup must be in service at the time of the switch request.

A primary data set may be replaced by another data set specified by DDNAME on the DDSCNTRL command provided that no DDSDCB opened for the duplicate data set exists at the time of the request. This would cause the new DDNAME to become the primary copy and the old primary to become the in-service backup copy.

The user may wish to verify that his primary and backup copies of a DDS are, in fact, the same. He may do this with the COMPARE operand of the DDSCNTRL command. To invoke this operation, he must be sure that a COMPRINT and DDSCMPIN DD card was supplied. When invoked, the

OS/VS1 utility IEBCOMPR is used to do the compare. To use this operand, LRECL must have been specified on the DDS DD cards for partitioned data sets.

See Section 3, entitled "DDS INITIALIZATION" for a description of DD card usage.

The status of the primary and backup DD names may be determined (in or out of service, which is primary, which is backup) by invoking DDSCNTRL with the STATUS operand.

The following is a list of restrictions and guidelines for using the DDS services.

1. All duplicate data sets must begin on a cylinder boundary and can have only one extent.

2. The user should be certain that any two data sets being declared as duplicates are, in fact, identical in their content.

3. Two tasks can reference the same DDSDCB provided it is treated as a serially reusable resource. In update mode the user must treat each READ-CHECK and WRITE-CHECK operation as a single function.

4. Only Disk resident data sets can be declared duplicates.

5. Only one DDS DCB per DDS can be opened at a time.

6. No copy and control functions can be used if the DDSDCB is opened for update (BPAM or BSAM).

7. DDS services are available only to the Special Real Time Operating System tasks.

8. Only 20 duplicate data set pairs are supported.

In the following example of typical use of DDS, the user wishes to create a duplicate BPAM data set and update an existing BSAM duplicate data set. The job step JCL would include these cards:

```
//BPAM1 DD DSN=BP1,DISP=(NEW,PASS),SPACE=(CYL,(1,,1)),UNIT=DISK
//BPAM2 DD DSN=BP2,DISP=(NEW,PASS),SPACE=(CYL,(1,,1)),UNIT=DISK
//BSAM1 DD DSN=BSM1,DISP=(OLD,PASS)
//BSAM2 DD DSN=BSM2,DISP=(OLD,PASS)
//DDSCTLIN DD *
      DDSNAMES     (BPAM1,BPAM2)
      DDSNAMES     (BSAM1,BSAM2)
/*
```

The OPEN and DDSDCB macros would be coded as follows:

```
DDSOPEN    (DDSBP1,(OUTPUT))
DDSOPEN    (DDSBSM1,(UPDAT))

DDSBP1     DDSDCB       DDNAME=BPAM1,...
DDSBSM1    DDSDCB       DDNAME=BSAM1,...
```

The READ, WRITE, and CHECK macros would be coded exactly as if they were standard OS.

The STOW and CLOSE macros would be coded as follows:

```
 DDSSTOW   DDSBP1,LIST,R
 LIST DC   CL8'MEMBER1',XL4'0'
 DDSCLOSE  (DDSBP1)
 DDSCLOSE  (DDSBSM1)
```

DDS FAILOVER/RESTART CONSIDERATIONS

The status of each declared DDS will be kept on a disk resident data set with DDNAME, DDSTATUS. All changes (via COPY and CONTROL) will be recorded on this data set. In the case of failover or restart, the status of each DDS will be taken from this data set.

If the user wishes to use an existing DDSTATUS for his declarations at initialization time, he must include in his DDSCTLIN input stream a REFRESH card as the first card (REFRESH can begin in any column except column 1). All the remaining cards (if any) will be ignored, and the declarations currently on the DDSTATUS data set will be used.

When initializing a backup computer, the first card in the DDSCTLIN input stream must be READONLY which may start in any column except column 1. This will inhibit all data transfer to disk by DDS until failover occurs and this machine becomes primary. READONLY implies REFRESH, so the current declarations on DDSTATUS will be used.

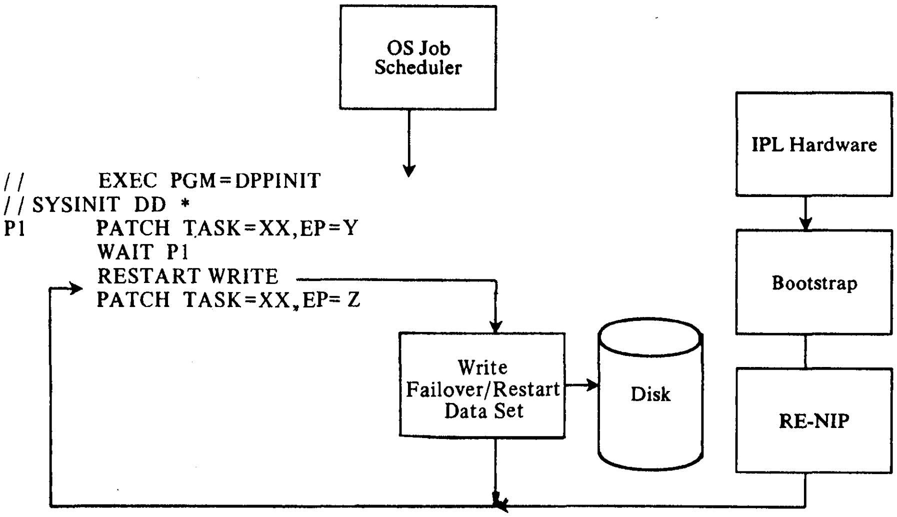
When DDS is entered during failover/restart, it expects all DDSDCB to be closed. Any task which has a DDSDCB opened at that time will be ABENDed with code 81 decimal (51 hex). Normal task clean up will then close the DDSDCB and free the associated storage gotten by DDS.

FAILOVER/RESTART FEATURE

The failover/restart feature of the Special Real Time Operating System is SYSGENable and optionally provides the continous monitor and probe function and the computer status panel.

Failover/restart operates by copying the contents of virtual storage, the OS/VS1 job queue, and the SWADS for the one or two partitions that encompass the realtime job, into a disk data set. If two-partition operation is being used, both SYSINIT streams must contain RESTART WRITE statements. The writing of the failover/restart data set is delayed until both partitions execute this statement. After this data set is written, a "bootstrap" record is written at the front of the data set, and the IPL1 and IPL2 records on the volume containing this data set are adjusted to allow them to read in the bootstrap program. Thus, the volume containing the failover/restart data set becomes an IPLable volume. IPLing this volume is the method of accomplishing the restart. If this occurs on a different CPU, the operation is known as a failover.

The effect of IPLing this volume is to return the System/370 to the identical state it was when the RESTART WRITE card was encountered in the SYSINIT Special Real Time Operating System initialization stream. This is illustrated in Figure 2-27. The failover/restart bootstrap restores virtual storage, the job queue data set, and one or two SWADS data sets to the identical state they were when the restart was written. ~~restart was written. No saving or restoring of the SYS1.SYSPOOL~~ No saving or restoring of the SYS1.SYSPOOL data sets occurs. Use of a scheduler work area (SWA) in place of SWADS by the MASTER or SLAVE partition will cause the SWADS not to be saved. The equivalent information is available within the partition.



```
//       EXEC PGM=DPPINIT
//SYSINIT DD *
P1       PATCH TASK=XX,EP=Y
         WAIT P1
         RESTART WRITE
         PATCH TASK=XX,EP= Z
```

Figure 2-27. Restart Process

Realtime jobs which use the failover/restart feature must observe the following restrictions:

1. The failover/restart data set and its copies must reside on a direct access volume. The volume may be on any device supported by OS/VS1. It may not be the volume containing the SYS1.NUCLEUS data set (OS IPL volume). No more than one failover/restart data set may be allocated on a volume. The failover/restart data must not be an OS temporary data set; it must reside entirely upon one volume and can contain only one extent. (Only the first extent will be used.)

2. The failover/restart data set should be allocated on a cylinder boundary. SYS1.SYSJOBQE and the SWADS data sets must end on a cylinder boundary.

3. The SWADS data sets cannot be temporary data sets unless SWA is used in place of SWADS.

4. No data set used by the realtime job should be a temporary data set nor should the realtime job be dependent on SYSIN data sets after the RESTART WRITE card is executed. Because the job is never ending, it should not use DD cards containing the SYSOUT parameter. If such SYSIN/SYSOUT data sets are used, contents may be lost. This does not apply to the SYSINIT input stream as it is read in its entirety prior to executing any of it.

5. No tape positioning is done by failover/restart.

6. At the time of restart read, all necessary direct access volumes must be mounted and ready. Data sets that are to be referenced and were allocated prior to restart write must exist on the same volume as they did prior to restart write. If DCBs were opened for any direct access data sets prior to restart write, these data sets must occupy exactly the same physical disk location they did prior to restart write. The device address upon which a particular volume resides may differ, however. A necessary volume is one that contains a system data set or that is allocated to the realtime job.

7. At the time of restart read, multiple volumes with the same volume serial number must not be accessible. There-NIP routine

will attempt to read the volume serial number from all direct access devices which were SYSGENed into the OS/VS1 system.

8. At the time of restart write, the user should take steps to ensure that no jobs are active in the CPU other than the realtime job and its SLAVE partition job, if any. If this restriction is ignored when the failover/restart data set is IPLed, these jobs will resume at the point where they were written without the benefit of a restored SWADS and possibly with data extent blocks (DEBs) containing invalid disk addresses. Resumption could occur in the middle of a DADSM function, thereby compromising VTOC and data set integrity.

9. Restrictions 3 through 8 do not apply if the failover/restart is to be written, but never read. Restrictions 1 and 2 apply in any case.

10. The Special Real Time Operating System initialization routine invokes restart when the RESTART WRITE card is encountered in the execution pass of the SYSINIT input stream. This is executed by issuing the WTFAILDS macro (no operands). A user program can also issue this macro. Use of this feature repetively (to take checkpoints) is not recommended for all the reasons listed above. In addition, since each execution of WTFAILDS would cause the existing copies of the failover/restart data set to be overlayed, a failover in the middle of the restart write could result in no usable failover/restart data set, old or new. Failover/restart is a method of getting a fast IPL; it is not a substitute for checkpoint restart.

11. If a failover/restart data set is to be written on one CPU and potentially read by any CPU other than the creating one, the following restrictions should be observed:

    a. The CPUs should have identical configuration or the OS/VS1 system involved should be a superset of all the CPUs.

    b. The CPUs should all be of the same model at the same EC/feature level to ensure that RMS will operate correctly.

    c. If the CPUs are of different real storage sizes, the failover/restart data set must be written by the one with the smallest real storage size. When IPLed on the larger CPU, this CPU's extra real storage will not be used.

12. A copy of a failover/restart data set can be made only by using IEHDASDR (an OS/VS1 utility) or DOMIRCPY (a Special Real Time Operating System utility to copy failover/restart data set). IEHDASDR can be used only in the sense of making a tape backup for later restore.

13. The WTFAILDS macro should not be used by an application program if the Time Drift Correction feature is used. This does not preclude the use of the RESTART WRITE statement in the DPPINIT input stream.

When failover/restart write is invoked, it copies all of virtual storage, SYS1.SYSJOBQE, and the SWADS data set(s) to the data set allocated by the DPPFAIL DD card. Copying of virtual storage consists of copying all real storage and those entries on the paging data set which are active. After the writes to DPPFAIL are completed, the desired backup copies of the entire failover/restart are made by copying from DPPFAIL to DPPFAILx, where x is a unique character (the method of copying the Failover data set is described later in this section). Each DPPFAILx must reside on a unique volume which is of the same device type as DPPFAIL. This operation has no connection with duplicate data set support and is independent of it.

Failover/restart data set write will not write the failover/restart data set if another realtime job (MASTER JOB) reached the Special Real Time Operating System initialization prior to the job issuing the WTFAILDS (or RESTART WRITE card). If this occurs, WTFAILDS will be treated as non-operative; although the pre-restart flag in SYSINIT PATCHes will be cleared. When the job having 'ownership' of failover/restart eligibility terminates, the next MASTER realtime job that starts will acquire it. In addition, if the byte at displacement X'0D' past the CSECT/ENTRY name DPPICINF in the OS/VS1 nucleus is nonzero, no job will acquire restart write eligibility. This byte is assembled as non-zero, but may be altered by using HMASPZAP, an OS/VS1 service aid.

(The name DPPICINF will be a CSECT name in the pageable nucleus in the Special Real Time Operating System without external interrupt handling; that is, without the TIMEEXT option or the CLOCKCP option on the VS SYSGEN macro and without the FAILEXT option on the FAILRST macro. In systems with external interrupt handling, it will be an ENTRY name in the CSECT IEAXYZ5.)

The return codes issued by WTFAILDS are listed below:

GR15	MEANING	Message Written to Routine Code 5
0	DPPFAIL data set written successfully or failover/restart write suppressed by other R/T job.	DPP090, DPP093
4	same return code as 0 except restart data set was just read.	DPP091
8	Invalid or missing DPPFAIL DD card.	DPP092
12_{10}	I/O error writing DPPFAIL or end of extent	DPP080-DPP087

When IPLing a failover/restart data set, several WAIT states can occur. They are listed below:

Code	Description
X'18'	CPU too small for failover/restart data set or other immediate program check in bootstrap.
X'19'	Program check in bootstrap while copying data sets or I/O error.
X'E2'	Machine check in restart bootstrap.
X'03'	One or more necessary disks missing.
Program check loop	Failover/restart data set has been scratched.
Hangs in LOAD	Failover/restart data set has been scratched or I/O error.

The load module DOMIRCPY is used internally by failover/restart write to copy DPPFAIL to DPPFAILx DD cards. It can also be invoked as a PATCHed task to perform the same operation in any realtime job, which is shown in the example below.

Return codes from DOMIRCPY

GR15	MEANING	Message to Routing Code 5
0	Copy Successful	None
8	One or more invalid DPPFAIL or DPPFAILx DD cards. No copy done.	DPP089
12	I/O error during COPY. COPY terminated.	DPP088

```
//X             JOB     1,1,MSGLEVEL=1
//                      EXEC PGM=DPPINIT
//STEPLIB       DD      ---
//SYSPRINT      DD      SYSOUT=A
//DPPFAIL       DD      DSN=FAILRST.DS,DISP=SHR       OLD F/RD/S
//DPPFAIL2      DD      DSN=FAILRST2.DS,DISP=OLD      NEW F/RD/S
//SYSINIT       DD      *
COPY    PATCH   EP=DOMIRCPY
      ` WAIT    COPY
        ABEND   0
//
```

Example 1

Continuous Monitor

The continuous monitor feature of the Special Real Time Operating System is available in all systems having the failover/restart feature. Its selection is made by coding the CONTMON parameter on the FAILRST SYSGEN macro.

The continuous monitor is started by PATCHing a task with EP=DOMIRCMN. This can be done by a user program, by a PATCH card in the SYSINIT input stream, or by the CMON parameter on the RESTART card. DOMIRCMN is a never ending program. DOMIRCMN should be invoked after the failover data set is written, if a failover/restart data set is being written.

The continuous monitor tests certain locations within the Special Real Time Operating System virtual storage data base on a periodic basis. The period is determined by the CONTINT parameter on the FAILRST macro.

If the continuous monitor determines that the test locations have not been modified for a certain period of time (indicating that some cyclic function has failed), it recommends that failover occur. The action taken depends upon the mode of operation of the continuous monitor, simplex or duplex. A system without a probe function generated (PROBE parameter in FAILRST macro) always operates in the simplex mode. In a system with the probe function, the continuous monitor operates in duplex mode unless one or both of the following conditions are met:

- The realtime job under which the continuous monitor is operating is not authorized to write a failover/restart data set (see Failover/Restart section).

- A probe function is running under any job on this CPU.

If either of these conditions is met, the continuous monitor will operate only in simplex mode. In the simplex mode, when the continuous monitor recommends failover, it issues message DPP098 and places the task under which it is executing in a permanent WAIT. In the duplex mode, the continuous monitor sends a signal via the direct control feature to the backup CPU that the continuous monitor is recommending failover. This signal is received by the probe function in the backup CPU. The backup CPU then initiates the failover.

The test locations examined by the continuous monitor are either determined implicitly by the options generated into the system, or additional customer test locations can be stated explicitly in the CONTADL parameter of the FAILRST SYSGEN macro. Each location to be tested is a 2-byte named, virtual storage resident data base item. This 2-byte item should be defined in an ITEM macro as initially zero. The The first byte of the item is the period in seconds at which the second byte will be updated. The second byte should be changed from 1 to 2, 3, ..., up to 250, back to 1 again at the rate specified in the first byte. If the value fails to change for a time interval equal to twice the first byte, the continuous monitor will recommend failover. If the second byte is zero, it indicates that the data base item will no longer be incremented at the rate specified and should not be checked again until it is once again nonzero. A value of 255 (hex FF) in the second byte indicates that the component is recommending failover. If this occurs, the If this occurs, the continuous monitor will recommend failover. Values of 251 through 254 are reserved for expansion.

Normally, the continuous monitor sends periodic signals to the probe function, but the converse does not occur. To have the continuous monitor report if the probe function is no longer checking it, the CMCKPRB parameter should be set to YES on the FAILRST SYSGEN macro.

Probe Function

The probe function is a SYSGENable option of the Special Real Time Operating System failover/restart feature. It operates in the backup CPU and tests the online CPU (the continuous monitor) via the direct control data bus. If the 4-bit value represented on the static signal lines fails to change for a time interval of twice the sample period (CONTINT parameter), the probe function recommends that the backup CPU become the prime CPU. The location of the 4-bit quantity on the static signal lines is determined by the PROBIT parameter of the FAILRST macro. These four lines must be connected between the two CPUs so that a signal placed on the lines by WRD in one CPU can be read by RDD in the other CPU and vice versa. A value of 15 (hex F) on the lines indicates that the continuous monitor is recommending failover to the probe function because the continuous monitor found a value of 255 in one of the data base items examined, or that the continuous monitor is recommending failover to the probe function because one of the test locations has failed to change at its specified rate. Thus the probe function will recommend failover when it gets a Continuous Monitor Recommended Failover signal or if the continuous monitor fails to change the bits on the static signal lines at the specified rate. (This could occur if the prime CPU went down.) In a system without the failover In a system without the failover confirmed external interrupt (FAILEXT parameter on the FAILRST macro), Failover Recommended is the same as Failover Confirmed (see Figure 2-28).

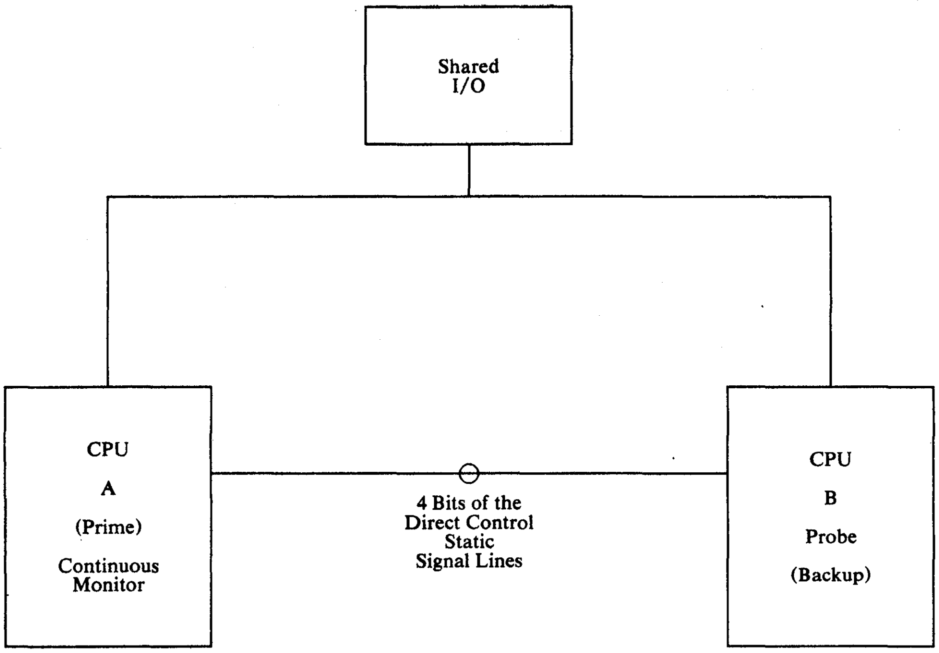


Figure 2-28. Probe Function Failure/Restart Feature

The probe function can be started in a realtime or non-realtime job. It will not start if another probe function is already operating on this CPU, or if a continuous monitor function operating in duplex mode is running. The probe can be started by the RESTART card in the Special Real Time Operating System SYSINIT stream or by a user program. It should be started on the backup CPU after the continuous monitor has been started on the primary CPU. If the probe is started first, it will immediately recommend failover.

The action taken by the probe when it enters the Failover Confirmed state is determined by its entry point. If the probe was entered at EP=DOMIRPRB, it will simulate a hardware IPL to the direct access device pointed to by the DPPFAIL DD card. This device should contain a successfully written failover/restart data set. If the probe was entered at EP=DOMIRPWT, it will return to its caller with a code of 4 in register 15. Coding the PROBE parameter in the RESTART card causes the DOMIRPWT entry to be used.

The DOMIRPRB entry point is intended for use in a duplex CPU environment where a system outage of 15 to 60 seconds can be tolerated. Upon reaching the Failover Confirmed state, DOMIRPRB will simulate a hardware IPL to the failover/restart data set. The realtime system will resume at a point after the execution of the RESTART WRITE card in the SYSINIT stream or the issuance of a WTFAILDS macro by an application program. The jobs, SYSIN and SYSOUT, and operating system running on the backup CPU prior to the simulated IPL will be lost.

The DOMIRPWT entry point is intended for use in duplex environments where a faster failover is needed. Using this scheme, the PROBE parameter is coded on the RESTART card. This causes the PROBE to be invoked after RESTART WRITE, if any. This causes a delay in the Special Real Time Operating System initialization until the probe function returns. Thus initialization stops until the probe enters the Failover Confirmed state. While the realtime job is executing only the probe, much of the remainder of it will be paged out by OS/VS1. Batch jobs can then be run. If the offline CPU later becomes the online CPU, these batch jobs can be cancelled if they are interfering with the realtime job. The following example depicts a sample SYSINIT stream for this type of operation.

```
 //     EXEC     PGM=DPPINIT

//SYSINIT        DD      *
        PATCH    EP=ONE,TASK=X         INIT TASKS
        PATCH    EP=TWO,TASK=Y
    RESTART      WRITE,PROBE,CMON      (IMPLIED WIAT ON ABOVE TWO)
        PATCH    EP=ONEONE,TASK=X
        PATCH    EP=TWOTWO,TASK=Y
/*
```

Remote System Reset

The Special Real Time Operating System supports the remote system reset RPQ (Z06741) in systems with the continuous monitor and probe function. This feature allows one CPU to force another CPU to execute a system reset. The probe function resets the online CPU which has just failed when it enters the Failover Confirmed state. Since during a failure the online CPU may have degraded to a disabled loop and has I/O devices reserved, this feature increases system availability by giving the backup CPU the ability to force a system reset in the online CPU. The RESET parameter in the FAILRST macro is used to include this feature in the probe function. The operand of the RESET parameter is a direct control signal-out line number (0-7) for the reset feature. Figure 2-29 depicts a two-CPU configuration with remote system reset.

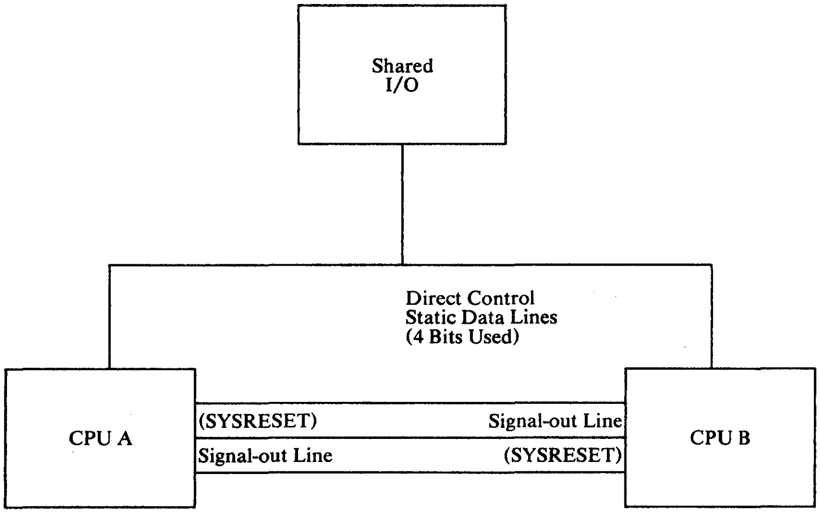


Figure 2-29. Remote System Reset Feature

Remote 2914 Switch

The Special Real Time Operating System also supports the automatic 2914 Remote Equipment Switch RPQs (880882 and 880920) in a system with the continuous monitor and probe functions. This feature allows devices which are connected to two CPUs through a 2914 switch to be automatically switched from the prime CPU to the backup CPU. When the probe function enters the Failover Confirmed state, it will cause the 2914 to switch the shared equipment to the backup CPU. The EQUIPSW parameter on the FAILRST macro specifies which direct control signal-out line (0-7) is to be used to cause the 2914 to switch shared equipment to the CPU issuing the direct control instruction. The EQUIPDY parameter specifies in milliseconds how long the probe function should delay after issuing the 2914 switch command until it returns (DOMIRPWT) or IPLs the failover/restart data set (DOMIRPRB). Figure 2-30 depicts a two-CPU configuration with 2914 remote switching. The remote system reset, 2914 Remote Switch, and computer status panel features are all independent of each other.

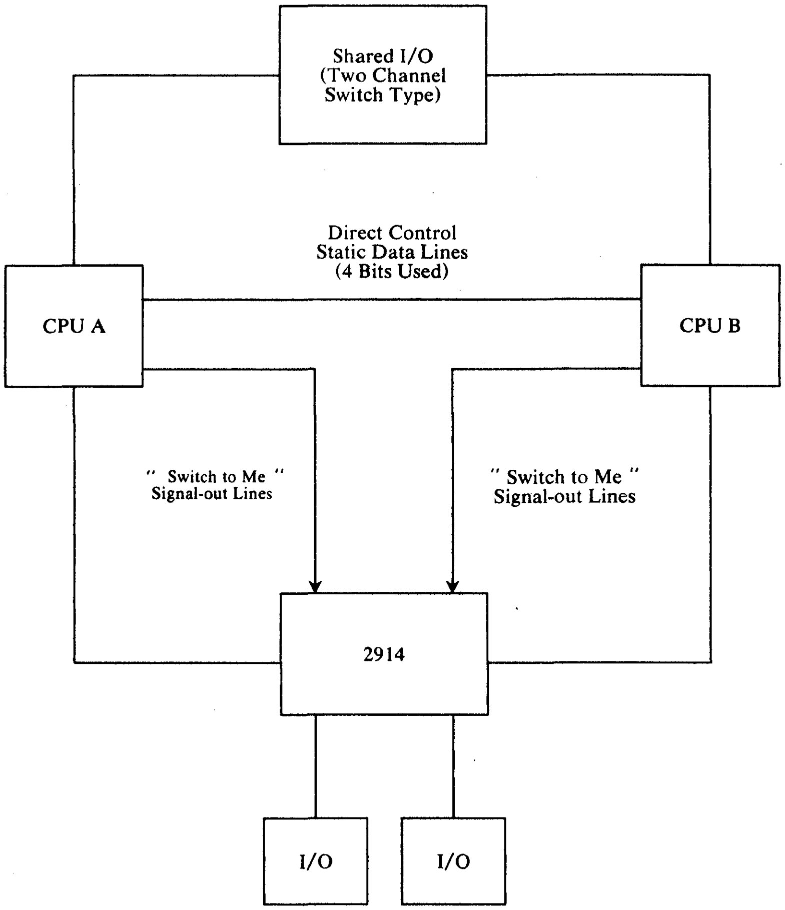


Figure 2-30. System With Automatic 2914 Switch

Computer Status Panel

The Special Real Time Operating System offers software for the computer status panel (see Figure 2-31) as an option for systems with the continuous monitor and probe features. In addition a smaller model can be supported for systems with only the continuous monitor feature.

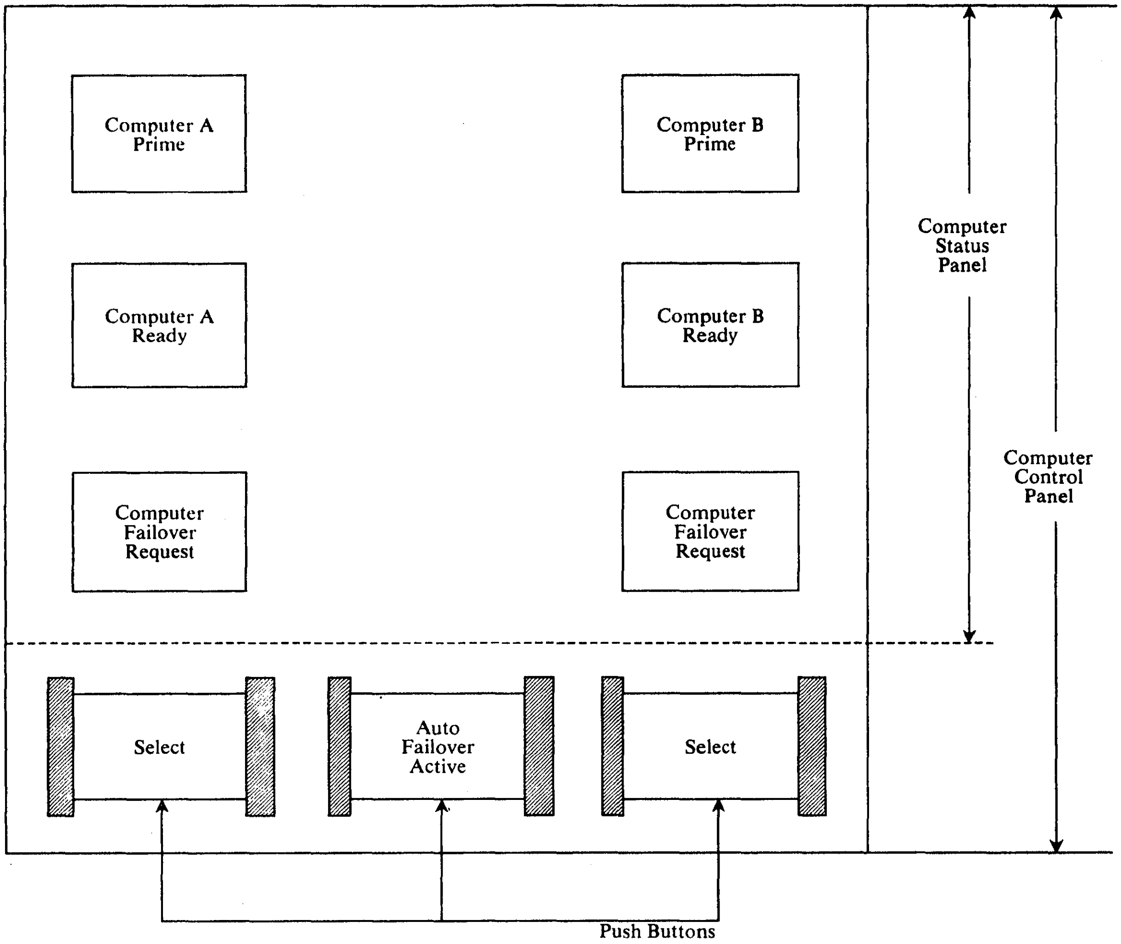


Figure 2-31. Computer Status Panel Indicators and Switches

The computer status panel consists of six indicator lights and two or three backlighted pushbuttons. The Computer A Prime/Computer B Prime lights are mutually exclusive; the one that is lit indicates which system is currently the online (prime) CPU. This light is illuminated by a pulse from a direct control signal-out line on the system that is the online system. It remains lit until a pulse is received on the same signal-out line on the other CPU, in which case the other PRIME light is lit. The Computer A Ready/Computer B Ready lights are illuminated by a signal on the direct control signal-out lines. They remain lit for approximately two seconds at which point they extinguish (time-out) unless they are re-illuminated prior to that time by another pulse on the same signal-out line. This light indicates that the CPU is successfully executing the online system or is capable of becoming the online system if it is the backup computer (probe function is running). For systems without a probe function, only these two indicators (four bulbs) are supported.

The Computer Failover Request light is illuminated by a bit on a direct control static data line. As long as the bit on the line is one, the light remains illuminated. Illumination is by the probe function when it enters a Failover Recommended state. The Select light backlighted pushbutton is lit by the probe function when it enters the Failover Confirmed state. This indication means that failover has begun. The Select and Failover Request lights are extinguished and the prime light is lit when the continuous monitor function starts to execute on the new failover-to-prime CPU.

In systems without the Failover Confirmed external interrupt (FAILEXT parameter on the FAILRST macro), the Failover Request and Select lights are illuminated simultaneously as the Failover Recommended and Failover Confirmed states are the same. In systems with the Failover Confirmed external interrupt, the Failover Request light will be illuminated when the probe function enters the Failover Recommended state. If the Auto Failover Active switch is on (is backlighted), an external signal interrupt occurs in the CPU lighting the Failover Request light. (Which external signal (Which external signal used (2-7) is indicated in the FAILEXT operand of the FAILRST macro.) This external interrupt causes the probe function to enter the Failover Confirmed state. If the Auto Failover Active switch is off, the SELECT backlight pushbutton must be pressed to cause the probe to enter the Failover Confirmed state. (The SELECT button causes the same external interrupt.)

If the continuous monitor begins to change the bit configuration on the four direct control static data lines, during the time the probe function is in the Failover Recommended state, the probe function extinguishes the Failover Request light and resumes normal operation.

In systems with the Failover Confirmed external interrupt feature, the SELECT pushbutton may be pressed at any time to force a failover. The SELECT pushbutton on the online computer may be pressed to force a restart. When the continuous monitor forces an IPL of the Failover/Restart data set, it places a special (14 hex E) bit configuration on the direct control static data lines to indicate that the probe function should delay one minute before continuing its checking of the online CPU.

The IPL that is forced by the continuous monitor is achieved by XCTLing to DOMIRIPL. This module exists in all systems with a probe function and may also be called by a user program (via CALL, LINK, XCTL, ATTACH, or PATCH) to force an IPL of the failover/restart data set.

The LTS parameter on the FAILRST macro is used to indicate which signal-out and static data lines are used to support the computer status panel. Figure 2-32 depicts a two-CPU configuration with a computer status panel.

The computer status panel is not an RPQ and must be fabricated by the customer.

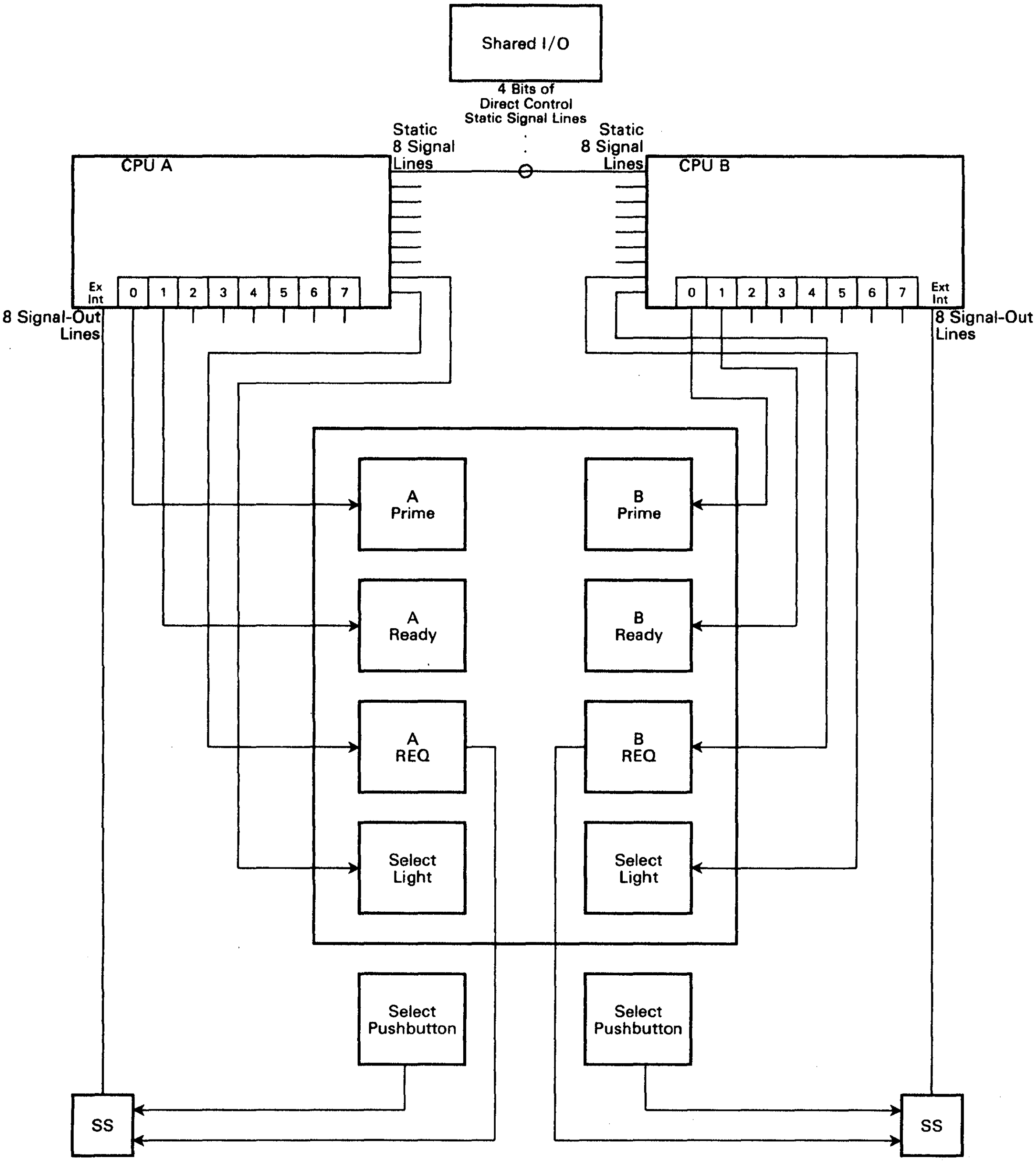


Figure 2-32. Computer Status Panel Connections (Functional)

ADDITIONAL SPECIAL REAL TIME OPERATING SYSTEM SERVICES

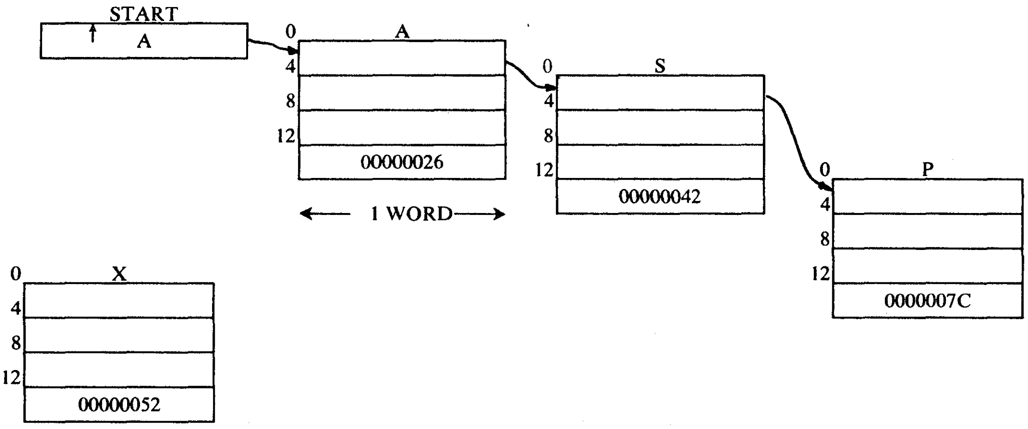
There are additional Special Real Time Operating System services that do not fall into the areas of task management or time management, etc. These additional services are:

CHAIN
GETWA/FREEWA
LOCK/DEFLOCK
PAGE FIX

CHAIN

CHAIN allows a programmer to modify a control block chain without the need of ENQ/DEQ to protect against another program modifying the chain at the same time. CHAIN operates as a Type I SVC. CHAIN can be used to add (ADD) a block (BLOCK=) to a specified chain (ORG=) or delete (REMOVE) a block from a chain. The block to be added (BLOCK=) may be placed at the start of the chain (POS=FIRST), the end of the chain (POS=LAST), or to put the block into the chain in a collating sequence (POS=disp). To place the block in collating sequence, the POS=disp specifies the displacement into the block to a word which is to be used to determine the block's relative position on the chain. This is shown below:

```
CHAIN          ADD,ORG=START,POS=12,BLOCK=X
```



In this example, block X will be added to the chain and will be inserted between blocks S and P.

If the blocks on the chain are not chained together by pointers in the first word of each block, the chaining field can be specified by INDEX= and supply the displacement into the blocks which are to be used for chaining pointers.

CHAIN will also post an ECB upon completion if the (ECB=) user requests this action.

All addresses are validity checked and must be within the partition (or either partition if two partition operation).

GETWA/FREEWA

The GETWA/FREEWA function provides the facility for obtaining short-term work areas without adversely increasing paging rates and without incurring all the overhead of a GETMAIN. The amount and sizes of GETWA areas are determined by the Special Real Time Operating System SYSGEN (VS Macro, GETWAS=) and may be changed at the Special Real Time Operating System initialization time by a GETWA card. The number of unique GETWA sizes is limited to 32. The maximum size of a GETWA area is limited to 30710 bytes. All sizes greater than 2048 bytes must be defined as a multiple of 2K. All must be defined as a multiple of 8 bytes.

Note: The Special Real Time Operating System requires a minimum GETWA size of 1024 bytes.

GETWA storage is requested via the GETWA macro and may be explicitly freed by FREEWA. The requestor may have the Special Real Time Operating System free the storage for him and thereby relieve himself of the necessity of keeping track of his GETWA storage. To have the Special Real Time Operating System free the gotten storage, he must use the TYPE= operand on his GETWA request. TYPE=AP requests the Special Real Time Operating System to release the GETWA storage when this PATCH completes. TYPE=AT frees TYPE=AT frees the storage when the task terminates (a DPATCH is issued). TYPE=PC is specified when the user wishes to free the storage explicitly with the FREEWA macro. Storage obtained with TYPE=PC will be lost to the system if the requesting task terminates and does not execute the FREEWA. Programs which are ATTACHed rather than PATCHed are defaulted to TYPE=PC and must explicitly release the area with a FREEWA macro.

The amount of space requested on a GETWA macro call all will be "rounded up" to the size of the smallest GETWA area defined which is as large or larger than the amount of space requested. (For example, sizes 8, 48, 96, and 1024 were generated and the request is for 680 bytes, the request will be satisfied with a 1024-byte block.)

When a GETWA is executed for a valid size area and all allocated blocks of that size are in use, the GETWA program will allocate additional space to satisfy the request. The additional The additional space will always be allocated in multiples of 2K and divided (if necessary) into blocks of a size that would otherwise be used to satisfy the request. The space, thus allocated, may be automatically released by a subsequent FREEWA when it is determined that there are sufficient free blocks of that size do not require immediate expansion again and an entire 2K block (or multiple) is not in use. This dynamic expansion of GETWA space will ensure that storage space is available as required. However, for performance considerations, the size of each GETWA area and the number of blocks defined for each size should approximate their actual usage during the realtime execution of an application. This will minimize both the CPU overhead and the amount of "wasted" storage areas.

GETWA storage area that has been obtained by one task may be passed to another task through the task management routines. The GETWA storage area may have been obtained originally via a GETWA macro call specifying TYPE=AP, AT, or PC and can be passed to another task by issuing a PATCH macro call of the form

```
  PATCH FREE=( ⎧AP⎫ , address),...
               ⎩AT⎭
```

where "address" is the address of the GETWA storage and "AP" or "AT" indicates the GETWA queue to which the GETWA storage is to be chained on the receiving task (i.e., PATCHed task).

An AP request causes the storage to be freed whenever the work queue element built in response to this PATCH is completed. An AT request causes the storage to be freed whenever the PATCHed task is terminated. In this respect, an AP or AT request is analogous to the PATCHed task acquiring the storage area by issuing a GETWA TYPE=AP or AT, respectively.

Note that if the PATCHing task receives a return code equal to or greater than 8 from the PATCH macro call, the PATCH cannot be executed, and the PATCHing task is responsible for freeing this GETWA storage area by executing a FREEWA macro call (eventhough the area may have originally been obtained by the PATCHing task through the issuance of a GETWA TYPE=AP or AT macro call).

If the PATCH is successful (return code less than 8), but the work queue built in response to the PATCH is later removed from the PATCHed tasks work queue chain before it can be executed, the storage area specified is freed by the Special Real Time Operating System when the work queue is purged eventhough the PATCH may have specified FREE= (AT, address). If the PATCH was successful, the PATCHing task must assume that the storage area passed has already been freed and no reference to that area should be made after the PATCH has been executed.

DEFLOCK/LOCK

The DEFLOCK/LOCK routines may be used in combination to define and reserve user specified resources without incurring the overhead of the ENQ/DEQ routines.

Each resource to be reserved must be defined to the Special Real Time Operating System by a DEFLOCK macro call (TYPE=GET). This macro call will cause a Special Real Time Operating System control block to be built describing the resource. The name of the resource will be returned in register 0, and the address of the new control block will be returned in register 1. This control block address must be specified whenever reserving a resource with a LOCK macro call. Once the control block has been defined, the address of the control block can be obtained by a DEFLOCK macro call (TYPE=FIND). After all processing for a particular resource has been completed, the control block may be released by a DEFLOCK macro call (TYPE=REL). Note that once the control block has been released, it must be redefined by a DEFLOCK macro call (TYPE=GET) before that resource can be reserved again by a LOCK macro call.

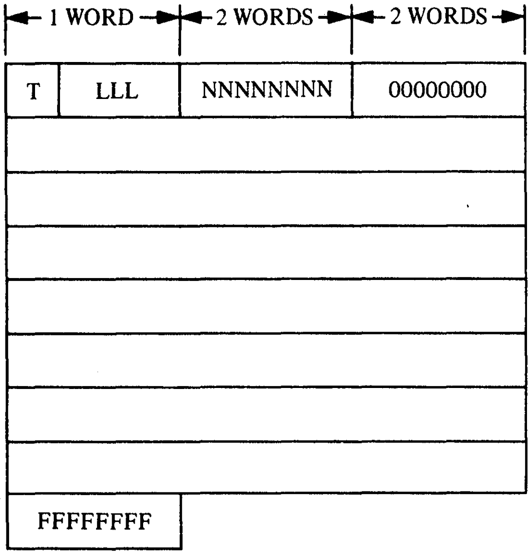
A LOCK macro call (TYPE=LOCK) is used to reserve exclusively a resource that has been released previously by a DEFLOCK macro call. If the resource is unavailable at the time the LOCK macro is executed, the requesting task is placed in a WAIT state until that resource becomes available. A LOCK macro call (TYPE=UNLOCK) is used to release control of the resource. Note that the LOCK macro call used to release the resource must be executed from the same task that executed the LOCK macro that reserved the resource. If a Special Real Time Operating System task (i.e., a PATCHed task) is DPATCHed or ABENDs before releasing the resource, the Special Real Time Operating System exit routine will release the resource for that task. However, if a non-Special Real Time Operating System task (i.e., an ATTACHed task) returns or ABENDs before releasing the resource, the LOCK will remain set indefinitely.

PAGE FIX

The Special Real Time Operating System provides a facility to allow users to 'fix' specific storage locations. To fix the storage, the user must create array DPPXFIX and put in it the names or numbers of arrays to be fixed and/or load modules to be fixed. Program DPPIPFIX will then process array DPPXFIX and LOAD the load modules and fix the virtual storage occupied by the specified arrays and load modules.

No attempt should be made to fix the storage for load modules which are link-edited as other than REENTRANT. The page fix function, when applied to load modules, operates by executing a LOAD macro to bring the module into storage and determine the address and length of the module. This LOAD is independent of the LOAD that is executed by the Special Real Time Operating Task Management or a LOAD, LINK or ATTACH executed by a user program. The result of this sequence is that if the module is not reentrant, each execution of the LOAD for a given module will bring into storage a separate copy of the module. Even though a non-reentrant load module may be fixed, the copy which is fixed cannot be accessed by normal means.

The format of the array named DPPXFIX is:



ARRAY DPPXFIX

where:

T = Type of fix request
L = load module
A = named array
N = numbered array
C = control block

LLL = Length of the fix request; zero indicates fix all storage occupied by the module or array.

NNNNNNNN = The left justified name of the load module or named array to be fixed or the array number of the numbered array to be fixed in the first word followed by a word of zeros. For control block requests, the left justified name must be 'CBGET' to indicate that CBGET storage is to be fixed; 'GETWA' to indicate that GETWA storage is to be fixed; or 'USERcccc' to indicate that a user control block is to be fixed.

0000000 = Each item must contain two words of zeros to be used by the page fix routine.

The following is an example of the control statements required to create a DPPXFIX array.

```
   #/DPPXUCTL                AREA=DBDEF,INPUT=*,OPTION=REPL
   ARRAY       NAME=DPPXFIX,INIT=YES

*          Request 1

A          ITEM TYPE=C,INIT='L'
B          ITEM TYPE=A,LEN=3,INIT=0
C          ITEM TYPE=C,INIT='DPPINIT1'
D          ITEM TYPE=F,RPT=2

*          Request 2

E          ITEM TYPE=C,INIT='N'
F          ITEM TYPE=A,LEN=3,INIT=50
G          ITEM TYPE=F,INIT=7
H          ITEM TYPE=F,RPT=2

*          Request 3

I          ITEM TYPE=C,INIT='A'
J          ITEM TYPE=A,LEN=3,INIT=150
K          ITEM TYPE=C,INIT='AA      '
L          ITEM TYPE=F,RPT=2

*          List Terminator

M          ITEM TYPE=F,INIT=-1

*          Request 4

M          ITEM TYPE=C,INIT='C'
N          ITEM TYPE=A,LEN=3,INIT=0
O          ITEM TYPE=C,INIT='CBGET'
P          ITEM TYPE=F,RPT=2

*          Request 5

Q          ITEM TYPE=C,INIT='C'
R          ITEM TYPE=A,LEN=3,NAME=ULEN
S          ITEM TYPE=C,INIT='USERXYZ1'
T          ITEM TYPE=F,NAME=USTART
U          ITEM TYPE=F,INIT=0
```

The above example creates an array named DPPXFIX for storage at initialization time and consists of:

1. A fix request for all (ITEM card B) of load module (ITEM card A) named 'DPPINIT1' (ITEM card C).

2. A fix request for 50 bytes (ITEM card F) of numbered array (ITEM card E) number 7 (ITEM card G).

3. A fix request for 150 bytes (ITEM card J) of named array (ITEM card I) named AA (ITEM card K).

4. A fix request for all (ITEM card N) CBGET storage (ITEM cards M and O).

5. A fix request for a user defined control block (ITEM cards Q and S). On this request, the final 4 characters of the name field (ITEM card S) must be 'USER'. The last 4 characters are not used by the page fix routine and may be used to further identify the control block by the user.

6. ITEM card M generates a fullword of binary ones to indicate the end of the array.

Note: On a user defined control block, the user must supply the length (ITEM card R) and the starting address (ITEM card T) before PATCHing the page fix routine (DPPIPFIX). By defining ITEM names for the length and start address, user PUTITEM macro calls could be used to fill in the array.

With array DPPXFIX created, the user must either have a PATCH card in his input stream for EP=DPPIPFIX or have a program which PATCHes DPPIPFIX.

It is also important to note that the user must terminate the realtime job step with a reply to the Input Message Processor (IMP) of the form

```
r  xx,CANCEL[,...]
```

in order to release the pages that have been "fixed" by the Special Real Time Operating System.

Note: It is recommended that all arrays being fixed should be created with a use count of one (USE=1), that the BNDRY parameter not be used and no other arrays be created with a use count of 1. Also, Array DPPXFIX must not be logged. If it is, a copy will be used which has non-zero for the two fullwords requested by page fix, and the page fix function will be bypassed.

TWO-PARTITION OPERATION

Two-partition operation may be requested at the Special Real Time Operating System SYSGEN time via the TWOPART operand on the VS macro. This allows programs running under control of the Special Real Time Operating System to communicate with programs running under control of the Special Real Time Operating System in a different job step. This environment is created by starting a job step and invoking the Special Real Time Operating System initialization (PGM=DPPINIT) in each of two partitions. Through this procedure, one of the job steps is designated as the MASTER and the other as a SLAVE to it. The MASTER job step has complete Special Real Time Operating System facilities included in it; the SLAVE has limited Special Real Time Operating System facilities in it, but has access to the facilities included in the MASTER partition.

The MASTER and SLAVE job steps are run in separate partitions of the OS/VS1 system and, as such, run under different storage protect keys, affording the user some protection for his data base, etc., from routines in the SLAVE partition. To attain effective communication between the two partitions, fetch protect must not be included in the system. This allows the programs in either partition to fetch data from the other partition, but prevents inadvertent storage of data into the other partition. Services are provided whereby programs in a SLAVE partition can store data into the data base, which is included in the MASTER partition.

Two-partition operation is initiated through normal Special Real Time Operating System initialization procedures with one additional control statement in each job's input stream. The MASTER job is designated by including the following control statement anywhere in the job's input stream.

```
 label     MASTER      SLAVE=jobname
```

where: Label is optional and must start in column 1

MASTER Specifies this is the MASTER job step

SLAVE= Specifies the name on the job card (JCL) of the job which is to be the SLAVE job step.

The SLAVE job is designated by including the following control statement any where in the SLAVE job's input stream.

```
label     SLAVE   MASTER=jobname
```

where: Label is option and must start in column 1

SLAVE Specifies this is the SLAVE job step

MASTER= Specifies the name on the job card (JCL) of the job which is to be the MASTER job step.

When initializing the system in this mode, both job steps must be started before the Special Real Time Operating System initialization can effectively proceed in either partition. When a RESTART WRITE statement is encountered in either partition, it must be included in both. The restart data sets are written only from the MASTER partition, but not before the SLAVE has completed the specified pre-restart processing.

When the MASTER partition terminates, the SLAVE is terminated with a USER 041 ABEND by the STAE routine. When the SLAVE terminates, two-partition operations are stopped in the MASTER until the SLAVE is restarted.

An attempt to start a SLAVE partition job when the MASTER job already has a SLAVE job executing will result in a user 41 ABEND for the second SLAVE job step.

Services not documented in the following two-partition description will exist in both the partitions.

Task Management

Both the MASTER and SLAVE partitions are provided Special Real Time Operating System task management services.

The PTN= parameter on the macro calls allows the user to specify the target partition for his PATCH, DPATCH and REPATCH:

```
                     ( OWN    )
                     { MASTER }
       PATCH ...,PTN={ SLAVE  }
                     ( FIND   )
```

If not explicitly specified on the macro call, the caller's own partition is the target partition for the macro.

Note on the PATCH macro:

a. The FREE= area must be in the partition, where the work represented by this PATCH will be executed and may not be used otherwise, because it is impossible to FREEMAIN storage in another than the own partition. For the same reason, FREE= is invalid if a PATCH goes to the other partition and REPATCH option is specified.

b. The PRTY/PRTYLOC parameter, if used, must specify the name of a task in the same partition as the created task.

While it is not a Special Real Time Operating System restriction, consideration must be given to passing data area addresses across partition boundaries. The area cannot be stored into except by programs in the same partition as the area or by supervisor services.

Task management control blocks will reside in both partitions as will the task management routines. However, if two-partition operation is desired, consideration should be given to the inclusion of certain task management routines in the Link Pack Area so that the same copy may be used for both partitions (see Coding and Performance Considerations).

Time Management

Both the MASTER and SLAVE partitions are provided Special Real Time Operating System time management services. The Special Real Time Operating System time and date are the same for both partitions.

The PTN= parameter on the PTIME macro allows the user to specify the partition in which a routine will be executed as the result of the PATCH macro call(s) by the time management routines.

The parameter

$$\text{PTIME ...,PTN=} \left\{ \begin{array}{l} \underline{\text{OWN}} \\ \text{SLAVE} \\ \text{MASTER} \\ \text{FIND} \end{array} \right\}$$

is identical in form and function with the PATCH parameter (PTN=) and is also subject to all restrictions specified for cross partition PATCHes.

Note that the time management routines and control blocks (PTQES) will reside only in the MASTER partition and, therefore, all resulting PATCHes will be issued from the MASTER partition.

Data Base

Both the MASTER and SLAVE partitions will be provided Special Real Time Operating System data base services. The Special Real Time Operating System data base will be the same for both partition. All data definition statements required to define the data base must be contained in the JCL for the MASTER partition job. The data definition statements relative to the data base are not required in the JCL for the SLAVE partition job and if present will be ignored. This includes the DD statements for the data base partitioned data set, all data base BDAM data sets, and all sequential data sets required for any DUMPLOG macro calls. Also, the DBREF statement, if desired, must be included in the input stream for the MASTER partition's job. The VS resident arrays and all data base control blocks will reside in the MASTER partition. Therefore, it should be noted that the user in the SLAVE partition cannot store directly into the VS resident data base but must use the Special Real Time Operating System data base macro calls.

All data base macro calls (i.e., GETLOG, PUTARRAY, etc.) will be supported from the SLAVE partition except GET/PUTARRAY with the ADDRLST option and GET/PUTITEM with the ADDRLST option. A return code 16 will be issued for these requests from the SLAVE partition.

Note: Data base macro calls issued from the SLAVE partition require the use of additional SVC routines to resolve the interpartition communication problem and, therefore, will incur additional system overhead.

The capability to define and reserve a resource will be provided for the SLAVE partition. However, partition LOCK/DEFLOCK routines will operate independently of each other; that is, a resource defined in one partition cannot be reserved via a LOCK macro call from the other partition. Cross-partition LOCK/DEFLOCK requests will not be supported.

Duplicate Data Set Support

Duplicate data set support is available to programs in both partitions. A data set pair which is DDSOPENed in one partition should not be accessed by programs in the other partition except by DDSOPENing it in both partitions.

Realtime Message Handler

All messages issued out of the SLAVE partition will be output from the MASTER partition. A message issued in the SLAVE partition will have an S affixed to the message when it is output.

EXAMPLE:

```
    DPP001S    2:29:23:3   01/FEB/74   REAL-TIME MESSAGE
```

The MSGDS DD card must be included in the JCL for each partition.

Input Message Processing

Input Message Processing (IMP) commands can be issued only to the MASTEI partition, but Input Message Processing can accept IMP commands to the SLAVE partition. The parameter SLAVE will follow the IMP command word. The parameters passed to the processing program will follow SLAVE.

EXAMPLE:

```
 'EXAMPLE,SLAVE,PARAM1,PARAM2,...PARAM20'
```

EXAMPLE: IMP Code.

SLAVE: Issue IMP command to SLAVE partition.

PARAM: Parameters accepted by the processing program.

Data Recording and Playback

Data recording and playback will run in both the MASTER and SLAVE partitions. The DRECOUT DD card must be included in the JCL for each partition. The DPBIN and SRTODUMP DD cards must be included in the JCL for each partition if Data Playback is to be run as a Special Real Time Operating System job.

Report Data Output Facility

The report data output facility will run in both the MASTER and SLAVE partitions.

SPECIAL REAL TIME OPERATING SYSTEM DEBUG GUIDE

User programs which run under the Special Real Time Operating System and ABEND will appear in a storage dump to be ABENDing in DPPTMON because the user programs ate loaded by DPPTPMON which then branches to them. As a result, the user programs are not represented by a PRB on the TCB's active RB chain. They run under the PRB for DPPTPMON. Figure 2-33 shows how the Special Real Time Operating System and OS control blocks would look upon entry to a user program.

The entry point of the program to which DPPTPMON gave control can be determined by looking 16_{10} bytes into the register save area pointed to by the TCBFSA field. Using the ABEND PSW and the entry point, the user can determine the displacement into the failing program of the last instruction executed prior to the failure. The program name can also be found by locating the LPRB with this entry point on the LPRB chain. An alternate method of locating the failing program is through the TCBUSER-TCBXCWQ-WQLCB-LCBEPAD and LCBEPNAM chain.

The TCBXNAME field of the TCBX will contain the task name (TASK=) of the task or DEPNDNT if this is a dependent task.

For additional debug information, refer to the IBM OS/VS1 Debugging Guide (GC24-5093).

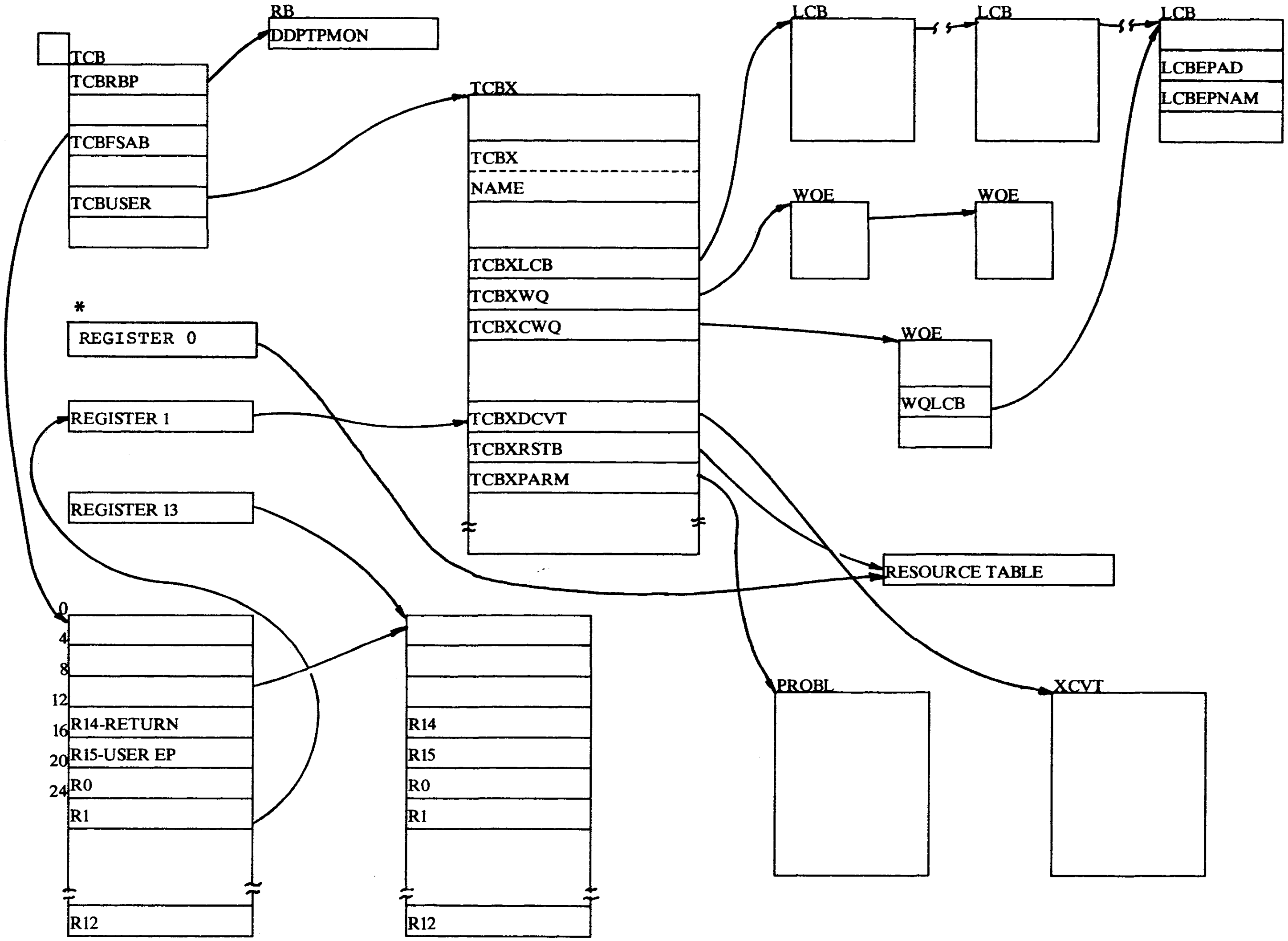


Figure 2-33. Control Block Format Entry to PATCHed Program

CODING AND PERFORMANCE CONSIDERATIONS

Certain considerations are normally made by a programmer when he is coding a program which will execute in a batch processing environment. However, additional considerations should be made when coding programs for a realtime environment. These additional considerations should be evaluated to determine if they will impact execution efficiency and system reliability. Some of these considerations are discussed on the following pages.

The Special Real Time Operating System tasks and user programs execute as subtasks; as a result, virtual storage gotten for a subtask by an OS/VS1 GETMAIN or REGMAIN must be explicitly freed. If it is not, the storage is lost and causes fragmentation within the partition. Storage which the Special Real Time Operating System GETWA routine gets for TYPE=PC must also be explicitly freed by FREEWA or this storage is also lost. OS/VS1 task management routines build control blocks with fixed PQA in a partition. If this PQA storage must be expanded during a realtime run, it may cause fragmentation of the partition. One consideration to help avoid fragmentation by PQA is to get the maximum number of TCBs which will be required for the realtime run through the use of the TCB control statement.

Programs coded for the Special Real Time Operating System should be coded as reentrant programs, if possible, to avoid having multiple copies in storage and to improve the OS/VS1 paging frequency. Also, because the Special Real Time Operating System PATCH monitor loads then branches to user programs, no user program which is to be PATCHed should ever issue EXIT (SVC3) or XCTL.

Careful consideration should be given to the amount of storage which is to be fixed by the page fix routine (DPPIPFIX), because page fixing has an adverse effect on the paging rate and may lead to thrashing.

Dependent Special Real Time Operating System tasks are initiated, execute once only, and are terminated. The additional overhead of initialization can be avoided by using independent tasks. This, however, causes the program (if reentrant) to remain in virtual storage and to maintain its resources. Careful consideration should therefore be given to whether user programs should be executed as dependent or independent tasks. Also, programmers who code programs to execute as independent tasks that open DCBs should be aware that the DCBs will not be closed after an execution of the program and that they will not be closed by a DPATCH. This may cause problems, since the TCB may be used by another Special Real Time Operating System task. This is also true of tasks which issue STIMER, then continue and never issue a TTIMER CANCEL for the STIMER.

If there is frequent use of GETWA blocks greater than 2K, the paging rate in the system may be improved if the user executes an OS/VS PGRLSE macro on the GETWA area prior to the FREEWA of the area.

Non-Special Real Time Operating System tasks (created by ATTACH) which reserve a resource via LOCK then ABEND, will not cause the Special Real Time Operating System ETXR routine to have the resource freed.

If the Special Real Time Operating System job step is to be terminated, careful consideration should be given to the method by which it is terminated, because the Special Real Time Operating System STAE routine will not be entered for ABEND codes 122, 13E, 222, 322, 522, or 722.

Programs which include the Special Real Time Operating System macro statements can be made more efficient and independent of the user SVC numbers generated by coding the DCVTR or DCVTLOC on Special Real Time Operating System macro statements.

T.e SYSGEN time interval for the Special Real Time Operating System cime update routine (PTIME= operand on the VS SYSGEN macro) should be made as large as practical in order to reduce the Special Real Time Operating System time management overhead. This is also true of data base logging overhead, which can be reduced by specifying as large as practical logging frequencies (LOGFREQ operand of LOG macro for the Special Real Time Operating System SYSGEN).

A logical BLOCK of a DA array will be represented by a physical block in the direct access data set. As a result, the DA array block size should be as large as possible for more efficient use of direct access storage.

Initialization using a DBREF NO statement will result in the loss of previously logged data.

Blocked arrays should be used whenever possible because they may then be used as either DA or VS resident arrays. overhead than named arrays when accessed through GETARRAY, PUTARRAY, GETBLOCK, or PUTBLOCK macros. Also, when a GETITEM or PUTITEM macro is executed, I/O processing is invoked to resolve item names to storage addresses. Therefore, it may be more efficient to resolve the item names at one time and to make subsequent accesses to the same items via these addresses rather than via the item names.

Programs should use data base macros to access the data base to ensure the integrity of the data base and to allow the program to be independent of the partition in which it executes (MASTER or SLAVE).

QPOS=DPATCH is not allowed if the task name represents a QH.

EP=(name DELETE) will remove the load module for the QP under which the work queue is processed and not necessarily for all QPs that may have processed work from this QH.

The TCBX address returned from the PATCH macro is the address of the QH TCBX. If the user is using this address to interrogate the progress of the work, the information that is picked up may not be meaningful.

PRTY= and QL= will never be meaningful when the NAME= specifies a QH. These parameters are established when the QH is defined.

If several PATCHes are executed with the PROBL or other work space to be freed by the FREE= parameter on the last PATCH, the last PATCH executed may not be the last to complete processing. The result may be that the area is freed before the last use of the area. This can happen when the PATCH is executed by the PTIME function.

An immediate DPATCH(DPATCH TYPE=I) to a QP will terminate the work currently active, but will not delete the task. All other DPATCHs for a QP and all DPATCH's for a QH are disallowed.

GETWA TYPE=AT executing under a QP or GETWA areas passed via PATCH to the AT chain of a QP will never be freed by Special Real Time Operating System (a QP can never be DPATCHed).

If the user were to set a LOCK while processing one Work Queue and return leaving the LOCK set intending the release the LOCK when processing the next work queue, it may not work because the next work queue may be processed under a different task (QP) and the LOCK must be released by the same task which it was set.

SPECIAL REAL TIME OPERATING SYSTEM ONLINE MACROS

For convenience, all online macros and their calling sequence are assembled in alphabetical order by macro name in the following pages.

BEGIN

The BEGIN macro provides standard OS linkage conventions for reentrant or non-reentrant routines. In general, the BEGIN statement is designed to:

- Identify and label the main control section or entry point address
- Save the calling program's general purpose registers.
- Establish main control section addressability.
- Prepare a save area.
- Define register usage through the EQUATE macro.

The following options are available for the BEGIN macro example:

Note: All register specifications should be absolute numbers rather than equated symbols (i.e., 13 rather than R13).

Programs using the BEGIN macro must be provided a higher save area by the calling program.

[symbol]	BEGIN	[csect name] [,ENTRY=symbol] [,BASE=(reg,[label])] [,ADDB=(reg 1, [reg 2,. . .,reg n])] [,SAVE={NONE (reg 1, reg 2)}] [,SAVEA={(symbol {GETMAIN GETWA} [,label])}] [,LV=number] [,SP=number] [ENTER=PATCH]

CSECT name
The name to be given to the main control section.

ENTRY=
The label name to be given to the first instruction and to be declared via the ENTRY assembler instruction.

BASE=
Specifies the general purpose register to be used as the initial main control section base and the label to be given to the point of zero displacement. Note that if no save area is requested and "reg" is omitted, register 15 is assumed. If a save area is to be assembled internally and a "reg" is omitted, register 13 is assumed. If a save area is to be obtained via a GETMAIN or GETWA and "reg" is omitted, register 12 is assumed.

ADDB=
Specifies additional main control section bases to be initialized at zero displacement + 4096, zero displacement + 2*4096, etc.

SAVE=
Indicates range of general purpose registers to be saved in the caller's save area. If omitted, registers 14 through 12 are saved.

SAVEA=
Specifies the type of save area to be used by the program. "SAVEA=GETMAIN" indicates that a GETMAIN is to be issued to obtain the save area. "SAVEA=GETWA" indicates that GETWA is to be issued to obtain the save area. "SAVEA=symbol" indicates that the save area is to be expanded within the program. Both SAVEA=GETMAIN and SAVEA=GETWA will result in reentrant code being generated. SAVEA=symbol is non-reentrant. Register 13 will contain the address of the save area. If omitted, no save area will be reserved. The "label" operand, if specified, will be used as the name of the DSECT describing the work area.

LV=
The length, in bytes, of the storage area to be obtained via a GETMAIN or GETWA. If SAVEA=GETMAIN or GETWA and LV= omitted, 72 bytes will be obtained. If GETWA is specified and the LV= value is greater than the largest GETWA size allocated, a system ABEND (probably 0C4) will occur within the code expanded by the BEGIN macro.

SP=
The number of the subpool from which the save area is to be obtained. If omitted, 0 is assumed and when executed under OS/VS1, the subpool specification will default to 0.

ENTER=PATCH
Specifies that the program is always entered via a PATCH interface. This operand is used only when SAVEA=GETWA is specified. If this operand is omitted or if anything other than PATCH is coded, it is assumed that the program may be entered by a linkage other than PATCH. Use of this parameter allows a smaller macro expansion.

CHAIN

The CHAIN macro provides the facility for allowing multiple tasks to modify the same control block chains without the necessity of the user issuing ENQ/DEQ or getting himself into a disabled state.

[symbol]	CHAIN	$\left[\begin{array}{l}\underline{\text{ADD}},\\ \text{REMOVE},\end{array}\right]$ $\text{ORG=}\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}$ $\text{,BLOCK=}\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}$ $\left[\text{,POS= }\left\{\begin{array}{l}\underline{\text{FIRST}}\\ \text{LAST}\\ \text{disp.}\end{array}\right\}\right]$ $\left[\text{,INDEX= }\left\{\begin{array}{c}\text{(r)}\\ \text{value}\end{array}\right\}\right]$ $\left[\text{,ECB= }(\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\},\ \left[\left\{\begin{array}{c}\text{(r)}\\ \text{condition code}\end{array}\right\}\right]\ )\right]$ $\left[\left\{\begin{array}{l}\text{,DCVTR= r}\\ \text{,DCVTLOC= }\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\end{array}\right\}\right]$ $\left[\text{,MF=L}\right]\ \left[\text{,MF=(E,}\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\})\right]$ where ‘r’ is a general purpose register, 2-12.

ADD
If the control block specified is to be added to the chain. This value is the default value.

REMOVE
If the control block is to be removed from the chain.

ORG=
Specifies the origin of the chain. This address must be in the origin of the chain and not in a previous control block in the chain. The reason is that the task could lose control, and the chain could be modified so that the "previous" control block address would no longer be valid when the task regained CPU control. However, since CHAIN executes disabled, this cannot occur when the origin of the chain is used. Any RX-type instruction address format is valid.

BLOCK=
Address of control block to be added or removed. An RX-type instruction address format is valid.

POS=
Only valid with ADD and specifies at which end of the chain to insert the block. FIRST implies the end of the chain nearest the origin. LAST implies the end of the chain farthest from the origin. 'disp' is the displacement into the block of a fullword containing a value to be used as a comparand. This value is used to insert the block in an increasing collating sequence relative to other blocks on the chain.

INDEX=
Specifies the offset in the control block to the chain point (fullword). Note that INDEX does not apply to the ORG address -- i.e., ORG always specifies the exact address of the fullword containing the address of the first control block. The index may be loaded in a register, r, and INDEX=(r) specified. If this parameter is omitted, the chain pointer is assumed to be the first word of the block.

ECB=
Specifies an ECB to be posted with the specified completion code after the chain is modified. Any RX-type insturction address format is valid for the ECB address. The condition code can be loaded in a register and specified as ECB=(addr,(r)).

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses that has the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

MF=
The list or execute forms of the CHAIN macro which are generated by specifying MF=L or MF=(E, address).

CHAIN Return Codes:

Decimal Code	Description
00	Successful completion.
04	REMOVE block address not found in chain. When this condition exists, the ECB will not be posted.
08	Invalid address in list. When this condition exists, the ECB will not be posted.
12	ECB specified had been previous posted.

DDSBLDD

The DDSBLDL macro is used to construct a note/point list of a DDS the same as BLDL for a standard OS data set. Return codes will be the same as from the OS BLDL macro.

[symbol]	DDSBLDL	(OS parameters)	[{,DCVTR=(r) ,DCVTLOC={(r) address}}]

The OS parameters are the same as in BLDL; that is, DCB address, list address.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses, having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

DDSCLOSE

The DDSCLOSE macro is used to close a DDSDCB. Only one DDSDCB can be specified and TYPE=T is not valid.

[symbol]	DDSCLOSE	(OS parameters) [{,DCVTR=(r) ,DCVTLOC={(r) address}}]

The valid OS parameters are DDSDCB address and MF=operand.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses, that has the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

DDSDCB

The DDSDCB macro is used to define the DCB for a Duplicate Data Set. The DDNAME specified in the macro should be the same as the name field of the DDSNAMES card in the DDSCTLIN input stream.

[symbol]	DDSDCB	(OS parameters)

This macro is coded identically to the OS DCB macro with the following notes:

- DSORG must be PS, PO, or DA.
- OPTCD can be omitted or W, R, or F.
- Multi-tracking cannot be specified.
- Only the following parameters are valid: BLKSIZE, DDNAME, DSORG, KEYLEN, LRECL, MACRF, NCP, OPTCD, RECFM, and SYNAD.

The DDSDCB macro performs the same function for the Duplicate Date Set (DDS) facility as the DCB macro performs forthe OS/VS1 data management. Whenever the address of a DCB is required with the other DDS macros, the address of a DDSDCB must be specified. The operands of DDSDCB are a subset of the DCB operands. The valid operands are listed below by access method. Refer to the *OS/VS1 Data Management Macro Instructions* manual (GC26-3793) for a detailed description of the operands. If an operand listed below does not have any restrictions other than those listed in the DDS description in Chapter 2, the field to the right of the operand is left blank. If the field is not blank, a restriction or extension to the OS/VS1 options is noted.

Operands valid with BDAM are:

- BLKSIZE=
- DDNAME=ddsname
- DSORG=DA only
- EODAD=
- KEYLEN=
- LRECL=

MACRF=

OPTCD=W/R, F only

RECFM=

SYNAD=

DEVD=DA

Operands valid with BSAM and BPAM are:

- BLKSIZE=
- DDNAME=ddsname
- DEVD=DA only
- DSORG=PS and PO
- EODAD
- KEYLEN=

LRECL=

MACRF=

NCP=

OPTCD=(W)

RECFM=

SYNAD=

Note: Invalid DCB options or operands must not be specified in the DD card DCB=operand.

DDSFIND

The DDSFIND macro is used to perform that same function as FIND but for a DDSDCB. Return codes will be the same as from the OS FIND macro.

[symbol]	DDSFIND	(OS parameters) [,DCVTR=(r) (r) ,DCVTLOC=address]

The OS parameters are the same as in OS FIND, that is, DCB address member/name/point list, type.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses that has the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

DDSOPEN

The DDSOPEN macro is used to open a DDSDCB. Only one Only one DDSDCB can be specified and, for update option, the task opening the DDSDCB must maintain exclusive control over it.

[symbol]	DDSOPEN	(OS parameters) $\left[\begin{Bmatrix}\text{,DCVTR= r}\\ \text{,DCVTLOC=}\begin{Bmatrix}\text{(r)}\\ \text{address}\end{Bmatrix}\end{Bmatrix}\right]$

The valid OS parameters are DDSDCB address, OPEN option and MF=.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses, having the address of a 4-byte core that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

DDSSTOW

The DDSSTOW macro is used to STOW a member of a partitional DDS. Return codes will be the same as from the OS STOW macro.

[symbol]	DDSSTOW	(OS parameters) [{ ,DCVTR= r ,DCVTLOC= { (r) address } }]

The OS parameters are the same as in OS STOW; that is, DCB address, member-name, and type.

DCVTR=r
 Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
 Where 'r' is the general purpose register (2-12) enclosed in parentheses, having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
 Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

DEFLOCK

Each resource to be reserved must be defined to the Special Real Time Operating System by the use of a DEFLOCK macro. The DEFLOCK macro will cause a control block to be built describing the resource. The name of the resource will be returned in register 0 and the address of the control block will be returned in register 1. This control block address must be used whenever reserving a resource with the LOCK macro. After all processing for a particular resource has been completed, the control block may be released by another DEFLOCK macro. Once the control block has been released, it must be re-defined by a DEFLOCK macro before that resource can be reserved again. In the case of two-partition operation, separate lock controls are maintained for each partition. Thus a program cannot use a lock control block created in the other partition.

The DEFLOCK macro may also be used to obtain the address of a previously defined lock control block.

The following operands are available for the DEFLOCK macro:

[symbol]	DEFLOCK	$\left\{\begin{matrix}\text{(r)}\\ \text{name}\end{matrix}\right\}$ $\left[\text{, TYPE=}\left\{\begin{matrix}\underline{\text{GET}}\\ \text{REL}\\ \text{FIND}\end{matrix}\right\}\right]$ $\left[\left\{\begin{matrix}\text{,DCVTR= r}\\ \\ \text{,DCVTLOC=}\left\{\begin{matrix}\text{(r)}\\ \text{address}\end{matrix}\right\}\end{matrix}\right\}\right]$

(r)
name
Is the positional operand that defines the 4-byte resource name. If (r) is specified, the general purpose register (0 or 2-12) contains the resource name.

TYPE=
Is used to indicate that a resource control block is being defined (TYPE=GET) or released (TYPE=REL). The address of the control block on TYPE=GET requests is returned in register 1.

A DEFLOCK with a TYPE=FIND option will cause the address of a previously defined lock control block to be returned in register 1.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses that has the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

When control is returned, register 15 contains one of the following return codes:

Decimal Code	Description
0	Successful completion.
4	Resource already defined. Register 1 contains the previously defined control block address.
8	Job step is not a Special Real Time Operating System task.
12	Resource not previously defined. (Valid only for TYPE=REL and TYPE=FIND requests.)

An independent task is created to be executed continuously over an indefinite time period. When an independent task is no longer required, it can be deleted by use of the DPATCH macro. Since the task may have several elements on its work queue an unconditional DPATCH does not allow these elements to be executed. Any ECBs associated with the work queue elements are posted with a DPATCH completion code. The DPATCH can be specified as W, which prevents losing any work queues, or it can be specified as conditional, which deletes the task only if it is dormant. A DPATCH immediate can be used to abnormally terminate a task that executes a long running program (e.g., report-routine). DPATCH of a queue holder is not allowed. Only TYPE=I or A is allowed to a queue processor.

```
[symbol]    DPATCH       [(r)  ]
                         [name ]

                         [        ( (r) ) ]
                         [        (  U  ) ]
                         [ ,TYPE= (  C  ) ]
                         [        (  W  ) ]
                         [        (  I  ) ]
                         [        (  A  ) ]

                         [       ( OWN    ) ]
                         [       ( MASTER ) ]
                         [ ,PTN= ( SLAVE  ) ]
                         [       ( FIND   ) ]

                        [( ,DCVTR=  (r)                )]
                        [(                              )]
                        [(              ( (r)     )    )]
                        [( ,DCVTLOC=    ( address )    )]

  Where 'r' is a general purpose register (2-12).
```

name
Is a 1 to 8 character name of the task to be deleted. If register form is specified, the register contains the address of the task name. If omitted, the current task is to be deleted.

TYPE=
If I is specified, the task is to be deleted immediately. It will be abnormally terminated with a user abend code of 65. However, a WQE that was queued with QPOS=DPATCH will still be executed as part of the cleanup processing.

If U is specified or the operand is omitted, the task specified is to be deleted unconditionally. Any work queue to the Any work queue to the task is posted as deleted. The current WQE, if executing, is allowed to complete. If C is specified, the specified task is deleted only if its work queue is currently empty. If W is specified, the task is deleted when the work queue becomes empty. This does not prevent work being queued to the task. If register form is specified, the register contains a numeric code of 0, 4, 8, or 12. A numeric code of 0 corresponds to a TYPE=U request, 4 corresponds to a TYPE=C request, 8 corresponds to a TYPE=W request, and 12 corresponds to a TYPE=I request. If A is specified, the program executing under the task will be abnormally terminated with use code 65. The task will not be deleted as an independent task and any work queues that are awaiting execution will not be deleted.

PTN=

In two-partition operation, this operand defines the target partition for the DPATCH. OWN means that the target partition is the partition that executes the DPATCH; MASTER defines the MASTER partition as the target partition. SLAVE defines the SLAVE partition as the target partition; if SLAVE is coded and two-partition operation is not initialized (no MASTER/SLAVE control cards in the SYSINIT input stream), the DPATCH will be rejected and a return code passed back in register 15. FIND causes the SVC to search for the task in its own partition first, then in the other partition and the first one found will be DPATCHed.

If register form is used to specify the task name and the PTN= operand is not specified, the high-order byte (byte 0) of the register also defines the target partition. The same bits are used as in the PATCH Supervisor list (SUPL), and they have the same meaning.

SUPLPTNS=1	PTN=SLAVE
SUPLPTNM=1	PTN=MASTER
both zero	PTN=OWN
both one	PTN=FIND

However, if the PTN= operand is specified, the expansion of the macro will insert the proper bit into the high-order byte.

DCVTR=r

Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)

Where 'r' is the general purpose register (2-12) enclosed in parentheses having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address

Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

When control is returned, register 15 contains one of the following return codes:

Decimal Code	Description
0	Successful completion.
4	Task was already DPATCHed=W
8	Task was already DPATCHed=U
12 No DPATCH	Task is not dormant (DPATCH=C only)
16 No DPATCH	Task is being removed
20 No DPATCH	Task name not found on independent task chain
22 No DPATCH	PTN=SLAVE requested but not initialized
24 No DPATCH	Invalid parameters passed.
28 No DPATCH	Task name specified a queue processor and type not I or A or task name specified a queue holder.

DPPXBLKS

The DPPXBLKS macro generates DSECTs for various Special Real Time Operating System and OS/VS1 control blocks. define requested control blocks. When a keyword is omitted, the When a keyword is omitted, the control block associated with that keyword is not expanded. With the exception of the "TYPE-" parameter, any non-blank character is acceptable as the keyword operand.

The following operands are available for DPPXBLKS.

[symbol]	DPPXBLKS	[TYPE= {DSECT CSECT}] [,REGS=] [,TASK=] [,TCBX=] [,TMCT=] [,WQ=] [,LCB=] [,GFCB=] [,GFMB=] [,TIME=] [,PTQE=] [,TIMED=] [,PTIMEL=] [,DDS=] [,DDSDA=] [,OS=] [,TCB=] [,CVT=] [,RB=] [,SRTOS=] [,XCVT=] [,SCVT=] [,SUPL=] [,REPL=] [,DB=] [,ALTPRI=] [,ALTSEC=] [,DADD=] [,DIRB=] [,DIRR=] [,DMPHDR=] [,PBT=] [,LOGCB=] [,LOGHDR=] [,DACNTL=] [LOCK=] [,MSG=] [,IMP=] [,DREC=] [,GFMB]

TYPE=
Used in conjunction with the TCBX, TMCT, WQ, LCB, GFCB, and PTQE parameters and indicates that the control block is to be expanded as a DSECT or CSECT. If omitted, DSECT is assumed.

REGS=
Indicates that the macro is to be expanded to provide register equates.

TASK=
Used to indicate that the control blocks related to task management are to be expanded as DSECTs (CSECTs). These are the TCBX, TMCT, WQ, LCB, and GFCB control block.

TCBX=
Used to indicate that the control block for the TCB extension (TCBX is to be expanded as a DSECT (CSECT).

TMCT=
Used to indicate that the control block for the task management control table (TMCT) is to be expanded as a DSECT (CSECT). The control block for the GETWA/FREEWA main block (GFMB) will also be expanded.

WQ=
Indicates that the control block for the work queue (WQ) is to be expanded as a DSECT (CSECT).

LCB=
Indicates that the control block for the load control block (LCB) is to be expanded as a DSECT (CSECT).

GFCB=
Indicates that the control block for the GETWA/FREEWA control block (GFCB and GFBE) are to be expanded as a DSECTs (CSECTs).

TIME=
Indicates that the control blocks PTQE, TIMED, and PTIMEL, related to time management are to be expanded as DSECTs (CSECTs).

PTQE=
Indicates that the control block for the PTIME queue element (PTQE) is to be expanded as a DSECT (CSECT).

TIMED=
Indicates that the time array DSECT (TIMED) is to be expanded. If "TIMED-PTIME" is specified, the control blocks used in time management are also expanded.

PTIMEL=
Indicates that the DSECT describing the PTIME input parameter list (PTIMEL) is to be expanded.

DDS=
Indicates that the DSECT (DDSDA) for duplicate data sets is to be expanded.

DDSDA=
Indicates that the DSECT describing the duplicate data set data area (DDSDA) is to be expanded.

OS=
Indicates that certain OS control blocks are to be expanded as DSECTs. These are the CVT, TCB, and RB control blocks.

CVT=
Indicates that the DSECT that describes the OS communications vector table (CVT) is to be expanded.

TCB=
Indicates that the DSECT that describes the OS Track Control Block (TCB) is to be expanded.

RB=
Indicates that the DSECT that describes the OS Request Block (RB) is to be expanded.

SRTOS=
Indicates that the Special Real Time Operating System communications table are to be expanded as DSECTs. These are the XCVT and SCVT tables.

XCVT=
Indicates that the DSECT that describes the primary Special Real Time Operating System communication table (XCVT) is to be expanded.

SCVT=
Indicates that the DSECT that describes the secondary Special Real Time Operating System communication table (SCVT) is to be expanded.

REPL=
Indicates that the DSECT that describes the PATCH supervisor input parameter list (SUPL) is to be expanded. The PATCH problem program input parameter list (PROBL) is also expanded.

SUPL=
Indicates that the DSECT that describes the PATCH supervisor input parameter list (SUPL) is to be expanded. The PATCH problem program input parameter list (PROBL) is also expanded.

DB=
Indicates that the control blocks for the data base are to be expanded as DSECTs. These are the ALTPRI, ALTSEC, DADD, DIRB, DIRR, DMPHDR, PBT, LOGCB, and LOGHDR.

ALTPRI=
Indicates that the control block for the primary array location table is to be expanded as a DSECT.

ALTSEC=
Indicates that the control block for the secondary array location table is to be expanded as a DSECT.

DACNTL=
Indicates that the control block that describes the direct access array control record is to be expanded as a DSECT

DADD=
Indicates that the control block for the direct access DD name table is to be expanded as a DSECT.

DIRB=
Indicates that the control block that describes the BLDL directory is to be expanded as a DSECT.

DIRR=
Indicates that the control block that describes the PDS directory is to be expanded as DSECT.

DMPHDR=
Indicates that the control block that describes the DUMPLOG header is to be expanded as a DSECT.

PBT=
Indicates that the control block that describes the Page Boundary Table is to be expanded as a DSECT.

LOGCB=
Indicates that the control block that describes the log control block is to be expanded as a DSECT.

LOGHDR=
Indicates that the control block that describes the LOG header is to be expanded as a DSECT.

LOCK=
Indicates that the control block for the LOCK control block (LOCKCBLK) is to be expanded as a DSECT.

MSG=
Indicates that the control block for the realtime message handler is to be expanded as a DSECT.

IMP=
Indicates that the DPPXIMP array, input message processing control block, is to be expanded as a DSECT.

DREC=
Indicates that the control block for data recording is to be expanded for a DSECT.

GFMB=
Indicates that the control block for the GETWA/FREEWA main block is to be expanded as a DSECT.

DUMPLOG

The DUMPLOG macro is used to dump (unload) logged copies of virtual storage resident arrays from the log data set to a sequential data set which may then be accessed by user-written routines.

[symbol]	DUMPLOG	{ NAME= name NUMBER= number NAMELST= { address (r) } NUMBLST= { address (r) } } [,STARTIM= { address (r) }] [,STOPTIM= { address (r) }] [,DUMPDD= { DUMPLOG ddname }] [,USRDATA= { address (r) }] [,DISP= { ADD NEW }] [{ ,DCVTR= r ,DCVTLOC= { address (r) } }] [,MF=L] [,MF= (E, { address (r) }]

The parameters NAME, NUMBER, NAMELST, and NUMBLST are mutually exclusive. The macro will not expand if more than one of these parameters is specified or if all of these parameters are omitted.

NAME=
Specifies the name of a named array for which the log array is to be dumped.

NUMBER=
Specifies the number assigned to a numbered array for which the log array is to be dumped.

NAMELST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of array names for which the log arrays are to be dumped. The name list will be a table of 8-byte entries with one valid array name in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the name list.

EXAMPLE: Name List

0	ARRAYNAM
8	HOUSTONƀ
16	TEXASƀƀƀ
24	X'FF'

NUMBLST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of array numbers for which the log array are to be dumped. The number list will be a table of halfword entries with one valid array number in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the number list.

EXAMPLE: Number List

0	H'1'
2	H'255'
4	H'139'
6	X'FF'

STARTIM=
Specifies an address or a register (2-12) containing the address of a 6-byte time and day field beginning on a fullword boundary. The first four bytes will contain a time in 10 millisecond units. The last two bytes will contain a binary value from 1 to 366 representing the day of the year. The logged copies of the array will be searched until a copy is found with a log time equal to or greater than the start time specified. If this parameter is omitted, dumping will commence with the oldest logged copy of the array.

STOPTIM=
Specifies an address or a register (2-12) containing the address of a 6-byte time and day field beginning on a fullword boundary. The first four bytes will contain a time in 10-millisecond units. The last two bytes will contain a binary value from 1 to 366 representing

the day of the year. The logged copies of the array will be dumped until the most recently logged copy has been dumped or until a copy is dumped with a log time equal to or greater than the stop time specified. If this parameter is omitted, dumping will terminate when the most recently logged copy of the array has been dumped.

Note: The DUMPLOG routine will insert a byte of X'FF' into the first byte of the logging header of the last copy of each array dumped to the sequential data set to indicate the end of the dump of each array is to be a user delog routine.

DUMPDD=
Specifies the name of a data definition statement which describes a sequential data set to receive the dumped copies of the array from the log data set. If this parameter is omitted, the DD name 'DUMPLOG' will be assumed as the default. The output will consist of spanned variable length records. The blocksize of the data set defined by the DUMPDD parameter must be at least 264 bytes but no more than 32760 bytes. The blocksize should be large enough to contain one array copy, the log header (24 bytes), the user dump header (256 bytes) if any, and the descriptor words for variable length records (8 bytes) for maximum processing efficiency.

USRDATA=
Specifies an address or a register (2-12) containing the address of a 256-byte area of user data to be contained in the dump header for each array on the sequential dump data set.

DISP=
Specifies whether the dumped copies are to be written at the beginning of the DUMPDD=data set (DISP=NEW) or added to the existing dumped copies (DISP=ADD).

If the disposition parameter specified on the DD statement for this data set is either OLD or SHR and the data set is empty then the first DUMPLOG request must specify DISP=NEW.

Specifying DISP=NEW on subsequent DUMPLOG requests will position a direct access data set to record one and will cause a tape data set to force the end of volume before the log copies are written.

DCVTR=
Specifies a register (2-12) which contains the address of the XCVT.

DCVTLOC=
Specifies the address or a register (2-12) enclosed in parentheses which contains the address of a core location which contains the address of the XCVT.

MF=L
Indicates that the list form of the macro is used to create a parameter list that can be referenced by an execute form of the DUMPLOG macro instruction.

MF=(E address

Specifies that the execute form of the DUMPLOG macro instruction and an existing parameter list are used.

Note: A zero returned in register 15 indicates successful corpletion. A non-zero in register 15 indicates that one or more errors were encountered during processing of this DUMPLOG request. The high-order byte of register 15 contains a count of the number of errors encountered and the low-order 3 bytes contain the address of the first invalid array name or number.

EXIT

The EXIT macro is to be used in conjunction with the BEGIN macro and will perform the exit linkage convention requirements. That is, register 13 will be restored to point to the caller's save area; the other general purpose registers that were saved will be restored; and the GETMAINed save area, if one exits, will be released.

The following options are available for the EXIT macro.

[symbol]	EXIT	$\left[\text{CODE} = \begin{Bmatrix}\text{number}\\ (15)\end{Bmatrix}\right]$ $\left[\text{,FREE} = \begin{Bmatrix}\underline{\text{YES}}\\ \text{NO}\end{Bmatrix}\right]$ $\left[\begin{Bmatrix}\text{,DCVTR= r}\\ \text{,DCVTLOC=}\begin{Bmatrix}\text{address}\\ \text{(r)}\end{Bmatrix}\end{Bmatrix}\right]$

CODE=
Specifies a return code to be passed to the RETURN macro; if a register is to contain the return code, only 15 is valid.

FREE=
Specifies whether or not EXIT is to FREE the save area allocated by the corresponding BEGIN macro. Either a FREEMAIN or FREEWA will be executed, depending upon how the save area was gotten.

The following parameters are meaningful only if FREE=YES is specified and the save area was allocated by GETWA.

DCVTR=
Specifies a register (2-12) which contains the address of the XCVT.

DCVTLOC=
Specifies the address or a register (2-12) enclosed in parentheses which contains the address of a storage location which contains the address of the XCVT.

FREEWA

The FREEWA macro releases control of a work area obtained via the GETWA macro. If the GETWA was not TYPE=PC, the FREEWA must be issued under the same task as the corresponding GETWA.

[symbol]	FREEWA	$\begin{Bmatrix}(r)\\ \text{address}\end{Bmatrix}$ $\begin{Bmatrix}\text{,DCVTR=(r)}\\ \text{,DCVTLOC=}\begin{Bmatrix}(r)\\ \text{(address)}\end{Bmatrix}\end{Bmatrix}$

where 'r' is a general purpose register (2-12)

If 'r' is specified, the register contains the address of the work area as returned to the caller after a GETWA macro execution.

If an 'address' is specified, it is a label of a fullword that contains the address of the work area as returned to the caller after a GETWA macro execution.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses having the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

When control is returned, register 15 contains one of the following return codes:

Decimal Code	Description
0	Successful completion.
4	Invalid FREEWA; • Area already free • Invalid address

GETARRAY

The GETARRAY macro is used to retrieve the data contained in one or more arrays of the data base, the address of those arrays in the data base, or the specifications of the items in the array(s). When data is to be retrieved from the data base and the amount of space required to contain the data is unknown, the GETARRAY macro with a TYPE=ADDR option can be used to obtain the size of the array before the macro with a TYPE=DATA or TYPE=SPEC option is used to retrieve the data. The macro does not actually return the size of the data area to contain an array's item names and specifications but instead will return the number of items contained in the array. The number of items can then be multiplied by 16 to obtain the actual size of the area for TYPE=SPEC. Where incr is specified, it may be any value from 1 to 255.

[symbol]	GETARRAY	$\left\{\begin{array}{l}\text{NUMBER=number, DATA=}\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\\ \text{NAME=name, DATA=}\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\\ \text{NAMELST= }(\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\text{ [,incr])}\\ \text{ADDRLST= }(\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\text{ [,incr])}\\ \text{NUMBLST= }(\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\text{ [,incr])}\end{array}\right\}$ $\left\{\begin{array}{l}\text{,DATALST= }(\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\text{[,incr])}\\ \text{,FINDLST= }(\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\text{[,incr])}\end{array}\right\}$ $\left[\text{,TYPE= }\left\{\begin{array}{c}\text{DATA}\\ \underline{\text{ADDR}}\\ \text{SPEC}\end{array}\right\}\right]$ $\left[\text{,PROTECT= }\left\{\begin{array}{c}\underline{\text{YES}}\\ \text{RISK}\end{array}\right\}\right]$ $\left[\left\{\begin{array}{l}\text{,DCVTR= r}\\ \text{,DCVTLOC=}\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\end{array}\right\}\right]$

The parameters NUMBER=, NAME=, NAMELST=, ADDRLST, NUMBLST, are mutually exclusive, only one may be specified.

NAME=
Is an 8-character name of a single array for which data is to be retrieved or the address is to be resolved.

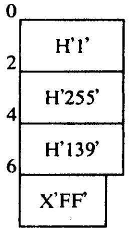
NUMBER=
Is an array number of a single array for which data is to be retrieved or the address is to be resolved.

DATA=
Is used with NAME= or NUMBER=. It specifies the address into which the content of the array is to be moved (TYPE=DATA) or the address, number of blocks, length of the array, or size of each block is to be moved. In the latter case, the address will occupy a fullword, and the number of blocks and the length of each block will occupy the next two halfwords.

NAMELST=
Specifies the address of a list of 8-character array names for which data is to be retrieved or the addresses are to be resolved. Incr is the value by which this address is to be incremented to locate the next name. If not specified, a value of 8 is assumed. A value or less than 9 must not be specified for incr. The list must be terminated by a byte containing X'FF' in the position that would be occupied by the first byte of the next name.

NUMBLST=
Specifies the address of a list of 2-byte fields containing array numbers for which data is to be retrieved or the addresses are to be resolved. Incr is the value by which this address is to be incremented to locate the next number. If incr is omitted, a value of two is assumed. A value less than two must not be specified for incr. The list must be terminated by a byte containing X'FF' in the first byte of the 2-byte field which would be occupied by the next array number.

EXAMPLE: Number List



ADDRLST=
Specifies the address of a list of data base array addresses as returned from a previous execution of this macro with NAME=, NAMELST=, or NUMBLST= specified and TYPE=ADDR. Incr is the value by which this address is to be incremented to locate the next array address. If incr is not specified, a value of 8 is assumed. This list must be terminated by four bytes containing X'FFFFFFFF' in the position that would be occupied by the first four bytes of the address of the next array. If the GETARRAY macro specifying NAMELST or NUMBLST is used to build this list, it will place the 4 X'FF' at the end of the list.

DATALST=
Is used with NAMELST= or NUMBLST= and TYPE=DATA or SPEC or ADDRLST= and TYPE=DATA. The address of a list of addresses into which the data from the specified arrays is to be moved. This must contain an entry

for each array for which data is to be retrieved. This entry will contain a fullword address which identifies the memory address to which the first byte of the array data is to be moved. Incr is the value by which the address within the list is to be incremented to pick up the memory address to receive the start of the next array.

If incr is not coded, a value of 4 is assumed.

FINDLST=
Is used with NAMELST= or NUMBLST= and TYPE=ADDR. It specifies the address of an area to receive an entry for each array specified. This entry will be eight bytes if incr is specified as less than 10 (or omitted) and 10 bytes if incr is specified as 10 or greater. The entry will be in the following format.

0	1	4	6	8
flag	array address	number blocks	array/bk size	number items

If bit 7 of the flag byte is set to 1, the array is direct access resident, and the address is not valid. If flag bit 6 is set to 1, it is a blocked array, and the block size must be multiplied by the number of blocks to determine the total size of the array. The number of items is the number of item names specified for this array through the offline definition of the array.

Incr is the value by which the address specified is to be incremented to determine the location to receive the next entry. If incr is not specified, a value of 8 is assumed.

TYPE=
Specifies the type of request. DATA specifies that the content of the array(s) is to be moved into the area specified by the DATA= or DATALST= parameter. ADDR specifies that the address of the array(s) and associated data as defined with the FINDLST parameter is to be moved into the area specified by DATA= (if NAME= or NUMBER= is specified) or FINDLST= (if NAMELST or NUMBLST is specified). SPEC specifies that the definition specifications as specified to the offline utility for each named item defined for the array(s) is to be moved into the area specified.

The data that is returned when the TYPE=SPEC is specified will contain a 16-byte entry for each named item in the following format:

name	len	type	disp	aid	rept	
0	8	9	10	12	14	16

name The 8-character name of the item.

len The length of the item in bytes.

type The data type of this item. An EBCDIC character as defined through the ITEM macro.

disp The displacement into the array (or block) of the item.

aid The ID of the array as assigned by the offline utility program.

rept The number of identical and sequential items defined by this entry.

PROTECT=<u>YES</u>
        RISK

If YES is specified, a LOCK will be set to prevent other programs that also specify PROTECT=YES from accessing the data base via the data base macros while the data is being moved. If another program is in the process of modifying the data base (a lock is set) when this macro is executed, this program will be delayed until the lock is released. The parameter has no effect if TYPE=ADDR or TYPE=SPEC is specified.

DCVTR=
Specifies a register which contains the address of the XCVT.

DCVTLOC=
Specifies the address or a register containing the location which contains the address of the XCVT.

After execution of the GETARRAY request, the return code in register 15 is set to zero to indicate successful completion or to four to indicate that the request could not be satisfied because of one or more of the following reasons:

- One or more of the named arrays is not defined to the system.
- A numbered array was requested which is higher than the highest numbered array defined to the system.
- A TYPE=DATA request was made for a direct access resident array.

The GETBLOCK macro will retrieve the data from blocked arrays and place that data into user-allocated storage. The macro may be used to retrieve one or more blocks of data from one or more arrays. The arrays may be either virtual storage or direct access resident.

[symbol]	GETBLOCK	$\left(\begin{array}{l}\text{NAME=}\begin{Bmatrix}\text{name}\\ \text{(r)}\end{Bmatrix}\\ \text{NUMBER=}\begin{Bmatrix}\text{number}\\ \text{(r)}\end{Bmatrix}\\ \text{NAMELST=}\begin{Bmatrix}\text{address}\\ \text{(r)}\end{Bmatrix}\\ \text{NUMBLST=}\begin{Bmatrix}\text{address}\\ \text{(r)}\end{Bmatrix}\end{array}\right)$ $\text{,DATALST=}\begin{Bmatrix}\text{address}\\ \text{(r)}\end{Bmatrix}$ $\left[\text{,PROTECT=}\begin{Bmatrix}\text{RISK}\\ \underline{\text{YES}}\end{Bmatrix}\right]$ $\left[\begin{Bmatrix}\text{,DCVTR= r}\\ \text{,DCVTLOC=}\begin{Bmatrix}\text{address}\\ \text{(r)}\end{Bmatrix}\end{Bmatrix}\right]$

The parameters NAME, NUMBER, NAMELST, and NUMBLST are mutually exclusive. The macro will not expand if more than one of these parameters is specified or if all of these parameters are omitted.

DCVTR=
Specifies a register (2-12) which contains the address of the XCVT.

DCVTLOC=
Specifies the address or a register (2-12) enclosed in parentheses which contains the address of a location which contains the address of the XCVT.

NAME=
Specifies the name or a register (2-12) from which data is to be retrieved.

NUMBER=
Specifies the number or a register (2-12) from which data is to be retrieved.

NAMELST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of array names from which data blocks are to be retrieved. The name list will be a table of 8-byte entries with one valid array name in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the name list.

EXAMPLE: Name List

Offset	Entry
0	ARRAYNAM
8	HOUSTONƀ
16	TEXASƀƀƀ
24	X'FF'

NUMBLST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of array numbers from which data blocks are to be retrieved. The number list will be a table of halfword entries with one valid array number in each entry. The first byte past the last entry will be set to X'FF' to indicate the end of the number list.

EXAMPLE: Number List

Offset	Entry
0	H'1'
2	H'255'
4	H'139'
6	X'FF'

DATALST=
Specifies the address or a register (not register 1) which contains the address of a user-constructed list of block numbers and of core addresses where the data blocks are to be written. The data list will be a table of 6-byte entries. Each entry will contain a 1-byte flag field, a 3-byte area address and a 2-byte block number.

PROTECT=
If YES is specified, a LOCK will be set to prevent other programs that also specify PROTECT=YES from accessing the data base while this GETBLOCK is in the process of accessing the data base. If RISK is specified, the data will be moved without regard to other programs which may be storing into the data base.

DATA LIST ENTRY DESCRIPTION:

0	1	2	3	4	5
FLAG BYTE	AREA ADDRESS			BLOCK NUMBER	

FLAG BYTE
X'40' -- Indicates the last entry to be processed for a particular entry in the name list or number list.

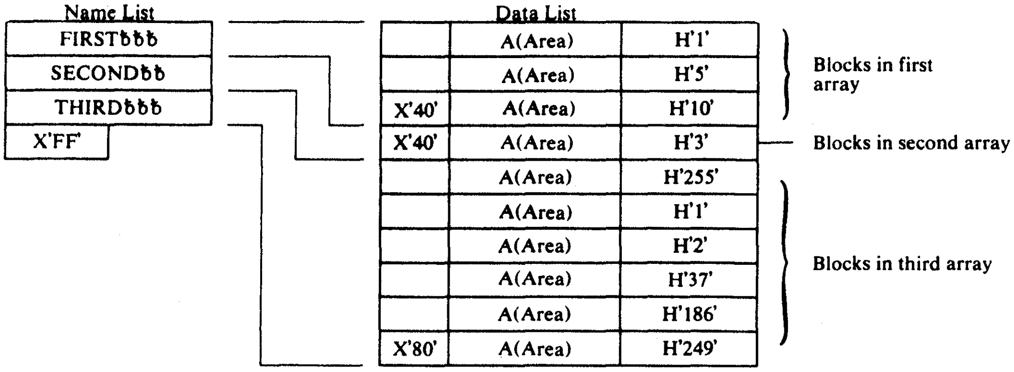
X'40' or X'80' -- Indicates the last entry in the data list.

AREA ADDRESS
The 3-byte address of a user-allocated area of storage where the data

block is to be written. The area must be large enough to contain the entire data block.

BLOCK NUMBER
The number assigned to the data block to be retrieved and placed in the area pointed to by the area address.

EXAMPLE: Data List and Name List



Name List

FIRSTƀƀƀ
SECONDƀƀ
THIRDƀƀƀ
X'FF'

Data List

	A(Area)	H'1'
	A(Area)	H'5'
X'40'	A(Area)	H'10'
X'40'	A(Area)	H'3'
	A(Area)	H'255'
	A(Area)	H'1'
	A(Area)	H'2'
	A(Area)	H'37'
	A(Area)	H'186'
X'80'	A(Area)	H'249'

Note: A zero returned in register 15 indicates successful completion. A non-zero returned in register 15 indicates that one or more errors were encountered during the processing of this GETBLOCK request. The high-order byte of register 15 contains a count of the number of errors encountered and the low-order three bytes contain the address of the first invalid array name or number.

The GETITEM macro is used to retrieve the data contained in one or more items of the data base or, alternately, the address or definition specification of those items in the data base. When data is to be retrieved from the data base and the amount of space required to contain the data is unknown, the GETITEM macro with a TYPE=ADDR option can be used to obtain the size of the item before the macro with a TYPE=DATA option is used to retrieve the data. Where incr is specified, it may be any value from 1 to 255.

```
[symbol]     GETITEM     ⎧NAME= name                                      ⎫
                         ⎪                  ⎧   (r)   ⎫                   ⎪
                         ⎨NAMELST=        ( ⎩ address ⎭ [,incr])          ⎬
                         ⎪                  ⎧   (r)   ⎫                   ⎪
                         ⎩ADDRLST=        ( ⎩ address ⎭ [,incr])          ⎭

                          ⎡         ⎧DATA⎫⎤
                          ⎢,TYPE=   ⎨ADDR⎬⎥
                          ⎣         ⎩SPEC⎭⎦

                          ⎡          ⎧  U   ⎫⎤
                          ⎣,BLKNO=   ⎩number⎭⎦

                                       ⎧   (r)   ⎫
                           ,DATA=    ( ⎩ address ⎭ [,incr])

                          ⎡            ⎧ YES ⎫⎤
                          ⎣,PROTECT=   ⎩RISK ⎭⎦

                         ⎡⎧,DCVTR= r                  ⎫⎤
                         ⎢⎨                ⎧  (r)   ⎫  ⎬⎥
                         ⎣⎩,DCVTLOC=       ⎩address ⎭  ⎭⎦
```

NAME=

Is an 8-character name of a single item for which data is to be retrieved or the address is to be resolved.

NAMELST=

Is the address of a list of 8-character ITEM names for which data is to be retrieved or the addresses to be resolved. Incr is the value by which this address is to be incremented to locate the next name. If not specified, a value of 8 is assumed. If specified, must not be less than 8. The end of the list must be indicated by a byte containing X'FF' in the position that would be occupied by the first byte of the next name.

If the items are contained in blocked arrays and TYPE=DATA or TYPE=ADDR is specified, the block number for which data is to be retrieved must be specified in the halfword immediately following the 8-byte name. Also the BLKNO= parameter should be specified as BLKNO=1, and the incr must be coded as a value of least 10.

TYPE=

Specifies the type of request. DATA specifies that the content of the ITEM(s) is to be returned. ADDR specifies that the address within the data base of the item(s) and the length(s) of the item(s) is to be returned. DATA and ADDR are invalid for direct access resident arrays. SPEC specifies that the definition specifications associated with each item is to be returned. If the ADDRLST parameter is used, this parameter must be omitted, or DATA must be specified.

ADDRLST=
Is the address of a list of data base item addresses as returned from a previous execution of this macro with NAME= or NAMELST= specified and TYPE=ADDR. Incr is the value by which this address is to be incremented to locate the next item address. If incr is not specified, a value of 4 is assumed. The end of the list must be indicated by a 4-byte field containing X'FFFFFFFF' in the position that would be occupied by the next address. If the GETITEM macro with NAMELST option is used to build this list, it will place the 4 X'FF' at the end of the list.

DATA=
Is the address into which the first data is to be stored. If TYPE=DATA was coded, DATA is the data from the first item specified, according to the length defined for the item in the data base. If TYP E=ADDR was coded, the length of the ITEM as defined in the data base is moved into the byte specified, and the address in the data base of the item is moved into the next three bytes. If TYPE=SPEC is coded, the specification for each item as defined to the offline utility will be returned. This will occupy an 8-byte field for each requested item and will have the following format:

len	type	disp	aid	rept
0	1	2	4	6

len The length of the item in bytes

type The data type of this item. An EBCDIC character as specified in the ITEM macro to the offline utility

disp The displacement into the array (or block) of this item

aid The ID of the array in which this item resides as assigned by the offline utility

rept The number of identical and sequential items defined by this entry.

Incr is the value by which the data address is to be incremented to determine the next address to move either the next address or data. If incr is not coded and TYPE=DATA, the length of the moved item will be used as the increment. If incr is not coded and TYPE=ADDR or TYPE=SPEC, a value of 4 is assumed. If incr is coded and TYPE=DATA, the data will be moved for the defined length of the item, up to the number of bytes defined by the incr value. If TYPE=ADDR or TYPE=SPEC is coded, four bytes of data will be moved in any case, and if the incr value is less than 4, the movement of data may overlay previously moved data.

When TYPE=ADDR is coded, a terminator flag (X'FFFFFFFF') will be moved into the position that would be occupied by the next address after the last to be resolved.

PROTECT=YES
RISK

If YES is specified, a LOCK will be set to prevent other programs that also specify PROTECT=YES from accessing the data base via the data base macros while the GETITEM is in the process of accessing the data base (a lock is set). When this macro is executed, the other program will be delayed until the lock is released. If RISK is specified, the data will be moved without regard to other programs which may be storing into the data base.

BLKNO=   U
number

If U is specified or if the parameter is omitted, the array is assumed to be unblocked.

A number is used to specify that the data is to be retrieved from a blocked array(s). If NAME= was specified, the number is the block number from which data is to be retrieved. If NAMELST= is specified, any number from 1 to 32765 may be coded to indicate that the block numbers are coded as part of the NAMELST.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses that has the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

When control is returned register 15 contains one of the following return codes:

Decimal Code	Description
0	Successful execution.
4	One or more of the item names specified could not be resolved or data was requested to be moved for an item with a defined length of 0 bytes.
8	Invalid options were passed to the GETITEM routine (probably the macro expansion had been modified).
12	A block number was specified for an unblocked array or a block number was specified that is greater than the highest block number defined for the array.
16	GETITEM request for an item that is contained in a direct access array.

The GETLOG macro retrieves logged arrays by time or by using array logging header information.

[symbol]	GETLOG	$\left\{\begin{matrix}\text{NAME=}\left\{\begin{matrix}\text{name}\\ \text{(r)}\end{matrix}\right\}\\ \text{NUMBER=}\left\{\begin{matrix}\text{number}\\ \text{(r)}\end{matrix}\right\}\end{matrix}\right\}$ $\text{,AREA=}\left\{\begin{matrix}\text{address}\\ \text{(r)}\end{matrix}\right\}$ $\left[\text{,STEP=}\left\{\begin{matrix}\underline{0}\\ \text{value}\\ \text{(r)}\end{matrix}\right\}\right]$ $\left[\left\{\begin{matrix}\text{,TIME=}\left\{\begin{matrix}\text{address}\\ \text{(r)}\end{matrix}\right\}\\ \text{,LOGHDR=}\left\{\begin{matrix}\text{address}\\ \text{(r)}\end{matrix}\right\}\end{matrix}\right.\right]$ $\left[\text{,PROTECT=}\left\{\begin{matrix}\text{YES}\\ \underline{\text{RISK}}\end{matrix}\right\}\right]$ $\left[\left\{\begin{matrix}\text{,DCVTR= r}\\ \text{,DCVTLOC=}\left\{\begin{matrix}\text{address}\\ \text{(r)}\end{matrix}\right\}\end{matrix}\right\}\right]$ [,MF=L] $\left[\text{,MF=(E,}\left\{\begin{matrix}\text{address}\\ \text{(r)}\end{matrix}\right\}\right]$

The parameters NAME and NUMBER are mutually exclusive. The macro will not expand if more than one of these parameters is specified or if both of these parameters are omitted.

NAME=
Specifies the name or a register (2-12) containing the address of the name of a named array for which a logged copy of the array is to be retrieved.

NUMBER=
Specifies the number or a register (2-12) containing the number of a numbered array for which a logged copy of the array is to be retrieved.

AREA=
Specifies an address or a register (2-12) containing the address of a user-allocated storage area where the logged copy of the array will be written upon retrieval from the log data set. This area must be large enough to hold the entire array and logheader (24 bytes). AREA is a required parameter.

STEP=
Is used to determine which copy of a logged array, relative to the TIME or LOGHDR parameters, will be retrieved from the log data set. The value is a signed number which may be either positive, negative, or zero. If the STEP parameter is omitted, a value of zero will be assumed as the default. If no sign is specified, the number is assumed to be positive.

PROTECT=
If YES is specified, a LOCK will be set to prevent other programs that specify PROTECT=YES from accessing the data base while this GETLOG is in the process of accessing the data base. If RISK is specified, the data will be moved without regard to other programs which may be storing into the data base.

(See GETLOG examples on the following pages for use of this parameter in combination with the TIME and LOGHDR parameters.)

TIME=
Specifies an address or a register (2-12) containing the address of a 6-byte time and day field beginning on a fullword boundary. The first four bytes will contain a time in 10 millisecond units. The last two bytes will contain a binary value from 1 to 366 representing the day of the year. This time and day will be used as a comparison value to establish a relative starting point to determine which copy of the array will be retrieved from the log data set. The TIME parameter cannot be specified if the LOGHDR parameter is specified.

LOGHDR=
Specifies an address or a register (2-12) containing the address of an array logging header. Information in this logging header will establish a relative starting point to determine which copy of the array will be retrieved from the log data set. The LOGHDR parameter cannot be specified if the TIME parameter is specified.

The logging header is a 24-byte control block which precedes the array, both as the array exists in VS and as it is written to the logging array. The logging header which was retrieved as part of a previous GETLOG macro can be used to retrieve additional data stepping either forward or backward in time.

DCVTR=
Specifies a register (2-12) which contains the address of the XCVT.

DCVTLOC=
Specifies the address or a register (2-12) enclosed in parentheses which contains the address of a location which contains the address of the XCVT.

MF=L
Indicates that the list form of the macro is used to create a parameter list that can be referenced by an execute form of the GETLOG macro instruction.

MF=(E, address)
   (r)
Specifies that the execute form of the GETLOG macro instruction and an existing parameter list are used.

When control is returned, register 15 contains one of the following return codes:

Decimal Code	Description
0	Successful completion.
4	Requested time is later than most recent logged copy. The most recently logged copy sill be read into the user defined area.
8	Invalid STEP parameter value. The number of log copies -1 will be substituted for the STEP parameter and that logged copy will be read into the user defined area.
16	The specified array had not been defined. No data is read into the user defined area.
20	The specified array is not a loggable array. No data is read into the user defined area.

GETLOG Examples

These examples will not describe the Assembler Language statement used to call the GETLOG macro, but will describe the response of the GETLOG routine to the different combinations of the TIME and LOGHDR parameters with the STEP parameter.

- STEP, TIME, and LOGHDR omitted -- Information will be extracted from the logging header on the virtual storage resident copy of the array, and the last logged copy of the array will be retrieved.

- TIME is specified and STEP= 0 or omitted -- An attempt will be made to retrieve a copy of the array logged at the exact time specified. If the array was not logged at that exact time, the first copy of the array logged after that time will be retrieved.

- TIME is specified and STEP=-2 -- The second copy of the array logged prior to the time specified will be retrieved.

- TIME is specified and STEP=+5 -- The fifth copy of the array logged after the time specified will be retrieved.

- LOGHDR is specified and STEP=0 or omitted -- Information will be extracted from the logging header specified and the copy represented by the logging header specified will be retrieved.

- LOGHDR is specified and STEP=-3 -- Information will be extracted from the logging header specified and the third copy prior to the copy represented by the logging header specified will be retrieved.

- LOGHDR is specified and STEP=+4 -- Information will be extracted from the logging header specified, and the fourth copy after the copy represented by the logging header specified will be retrieved.

GETWA

The GETWA macro provides the facility for obtaining, short-term work areas without adversely increasing paging rates. The work areas can be freed explicitly with the FREEWA macro or freed automatically at the end of the current patch queue or at the end of the current task processing. The address of the work area is returned in register 1. If the GETWA was unsuccessful, register 1 will contain 32 binary ones.

[symbol]	GETWA	$\left\{\begin{matrix}\text{(r)}\\ \text{length}\end{matrix}\right\}\left[\text{,TYPE=}\left\{\begin{matrix}\underline{\text{AP}}\\ \text{AT}\\ \text{PC}\end{matrix}\right\}\right]$ $\left[\left\{\begin{matrix}\text{,DCVTR= r}\\ \text{,DCVTLOC=}\left\{\begin{matrix}\text{(r)}\\ \text{address}\end{matrix}\right\}\end{matrix}\right\}\right]$
where 'r' is a general purpose register, 2-12.		

length
Is the length of the requested work area that can be specified in any RX-type format or in a general purpose register.

TYPE=
Specifies the status of the work area. If omitted or if TYPE=AP is specified, the work area will be freed automatically when the processing of the current PATCH work queue element is completed. If TYPE=AT is specified, the work area is freed when the current task terminates. If TYPE=PC is specified and the work area is completely under program control, a FREEWA must be specified for the work area. A FREEWA may be specified for any type of GETWA. GETWA TYPE=AT executing under a QP or GETWA areas passed via PATCH to the AT chain of a QP will never be freed by Special Real Time Operating System (a QP can never be DPATCHed).

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses, having the address of a 4-byte location that contains the address of XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

GETWA RETURN CODES:

Decimal Code	Description
0	Successful completion.
4	Invalid size requested.
12	An attempt was made to obtain additional GETWA storage. The attempt was unsuccessful because there was

insufficient CBGET storage to build the GETWA control blocks.

Every resource that has been previously defined by a DEFLOCK macro can be exclusively reserved by use of a LOCK macro. The address of the control block (which is returned by the DEFLOCK macro) must be specified in the LOCK macro. If the resource is unavailable at the time, the LOCK macro is issued, and the requesting task is placed in a wait state until that resource becomes available. Another LOCK macro must be used to release the resource.

Note: The LOCK macro used to release the resource must be executed from the same task as the LOCK macro used to reserve the resource.

If a Special Real Time Operating System task (i.e., a PATCHed task) terminates or ABENDs before releasing the resource, the Special Real Time Operating System exit routine will release the resource for that task. However, if a non-Special Real Time Operating System task (i.e., an ATTACHed task) returns or ABENDs before releasing the resource, the LOCK will remain set indefinitely.

The following operands are available for the LOCK macro:

[symbol]	LOCK	$\left\{\begin{matrix}\text{(r)}\\ \text{name}\end{matrix}\right\}$,CBLOC= $\left\{\begin{matrix}\text{(r)}\\ \text{address}\end{matrix}\right\}$ $\left[\text{,TYPE=}\left\{\begin{matrix}\text{LOCK}\\ \underline{\text{UNLOCK}}\end{matrix}\right\}\right]$ $\left[\left\{\begin{matrix}\text{,DCVTR= r}\\ \text{,DCVTLOC=}\left\{\begin{matrix}\text{(r)}\\ \text{address}\end{matrix}\right\}\end{matrix}\right\}\right]$

(r)
name
The positional operand defines the 4-byte resource name. If (r) is specified, the general purpose register must contain the resource name.

CBLOC=
The CBLOC= keyword parameter is used to indicate the address of the control block as defined by the DEFLOCK macro. If (r) is specified, general purpose register 1 should contain the control block address. "Address" is the label of a fullword that contains the control block address.

TYPE=
Is used to indicate that a resource is being reserved (TYPE=LOCK) or released (TYPE=UNLOCK).

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in

parentheses that has the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

When control is returned, register 15 contains one of the following return codes:

Decimal Code	Description
0	Successful completion.
4	This task has previously reserved and has control of the resource (TYPE=LOCK).
8	This task has not previously reserved the resource (TYPE=UNLOCK).
16	Invalid resource name and control block address combination. Resource will not be reserved.

MESSAGE

The MESSAGE macro is used by the Special Real Time Operating System and the programs running under the Special Real Time Operating System are used to cause a predefined message to be printed or displayed. The message must have been defined through the offline utility system using the DEFMSG macro.

[symbol]	MESSAGE	{(r) | number} [,ACT= {I | A | D}] [,ROUTE= ({(r) | C1} [,{(r) | C2}...,{(r) | C8}])] [,VAR= ({(r) | address1} [,{(r) | address2}...,{(r) | address]0}])] [,AREA= {(r) | address}] [,WAIT= {NO | YES}] [{,DCVTR= r | ,DCVTLOC= {(r) | address}}]

symbol
Is any symbol valid in the assembler language.

number
Is a unique 3-digit number which identifies the requested message. (r) is the general purpose register (2-12) enclosed in parentheses which contains the message number.

ACT=
Is the action code to be appended to the message number. I denotes information. A denotes action is required, and D denotes that a decision is required. If not coded, the action code specified through the offline utility will be used.

ROUTE=
Is the code or codes that identify the devices on which this message is to be displayed or printed. Unique routing codes are associated with de~ices during the Special Real Time Operating System build procedure. If this parameter is not included, the message will be routed to the devices specified through the offline utility procedure using the DEFMSG macro. The maximum number of routing codes that can be specified is 8. (r) is the general purpose register (2-12) enclosed in parentheses which contains the route code.

VAR=
Is variable data to be converted and inserted into the message to be output. The variables must be in the sequence and lengths specified through the offline definition of the message. The maximum number of variables that can be specified is 10, addr is any address valid in an RX-type instruction, and (r) is a general purpose register (2-12) which contains the address.

WAIT=
Informs the Special Real Time Operating System that performance of the active task cannot continue until the specified message has been

issued. YES specifies that the active task is to go into a wait state; and NO specifies that the active task is not to wait until the specified message has been issued.

AREA=
Is the address into which the formatted message is to be returned to the caller. The term addr is any address valid in an RX-type instruction and (r) is any register that was loaded with the address.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where (r) is the general purpose register (2-12) enclosed in parentheses that has the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

MESSAGE RETURN CODES:

The Message Handler will issue return codes through register 15. If the return code is 08 or greater, the message is not output.

Decimal Code	Description
0	Normal completion.
2	Specified number of variables less than number of variables specified through offline utility procedure. The remaining variables are padded with hyphens.
4	Specified number of variables greater than number of variables specified through offline utility procudure. The last variables in the list are dropped until number of specified variables equals number of variables defined through offline utility procedure.
8	Invalid message number.
12	Invalid routing code.
16	Input/output error.

MESSAGE MACRO -- List Form

The list form of the MESSAGE macro is used to construct a problem program parameter list. This problem program parameter list can be referred to in the execute form of a MESSAGE macro instruction.

The description of the standard form of the MESSAGE macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only. The message must have been defined through the offline utility system using the DEFMSG macro. The message text will be truncated to conform to the length restrictions of the device(s) it will be routed to.

[symbol]	MESSAGE	number [,ACT= {I | A | D}] [,ROUTE= (C1,[C2,...,C8])] [,VAR= (addr1,[addr2,...,addr10]] ,MF=L [,WAIT= {NO | YES}]

symbol
 Is any symbol valid in the Assembler Language.

addr
 Is any address that may be written in an A-type address constant.

ROUTE=
 The MESSAGE macro will expand space for eight route codes if none are specified.

MF=L
 Indicates the list form of the MESSAGE macro instruction.

MESSAGE MACRO -- Execute Form

A remote control program parameter list is used in, and can be modified by, the execute form of the MESSAGE macro instruction. The control program parameter list is generated by the list form of the MESSAGE macro instruction.

The description of the standard form of the MESSAGE macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execution forms. The format description below indicates the optional and required operands in the execute form only. The message must have been defined through the offline utility system using DEFMSG macro. The message text will be truncated to conform to the length restrictions of the device(s) to which it will be routed.

```
[symbol]  MESSAGE  {  (r)  } [,ACT= {I}][,ROUTE=({(r)} [, {(r)} ..., {(r)}])]
                   {number}        {A}           {c1 }     {c2 }       {c8 }
                                   {D}

                   [,VAR= ({  (r)    } ,[{  (r)    } ,...., {  (r)     }])]
                          {address1}    {address2}        {address10}

                   [,AREA= { (r)}]
                           {addr}

                   [,MF=  E, {remote list address} ]
                             {        (r)        }

                   [,WAIT= {NO }][ {,DCVTR=    r        } ]
                           {YES}   {,DCVTLOC= {  (r)   }}
                                              {address}
```

symbol
Is any symbol valid in the Assembler Language.

addr
Is any address that can be written in an A-type address constant.

MF=(E,remote list address)
Indicates the execute form of the MESSAGE macro instruction using a remote parameter list. The address of the remote parameter list can be loaded into register 1, in which case MF=(E,(1)) should be coded.

ROUTE=
If more route codes are expanded in the remote list than are required in the MF=E form, only the number specified on the MF=E form will be modified.' For example, if the remote list expanded 5 route codes and the MF=E only 2, only the first 2 in the remote list would be modified and the message would be routed to all 5 route codes. If this is not desired, the remote list should be zeroed prior to executing the MF=E form.

PATCH

The PATCH macro is used to create a Special Real Time Operating System task and to queue work to it. If no task name is specified, a dependent task will be created. A dependent task can accept only one PATCH and flags are set which will cause it to disappear as soon as the work represented by the single queue element is completed. If a name is specified, the PATCH SVC checks to determine if an independent task by that name already exists, and if not, an independent task is created. Then the work is queued to that task. Independent tasks will be kept available until they are removed from the system explicitly via the DPATCH macro; if all queued work is completed, they will go into a dormant state, ready to accept more work with function. the next PATCH.

The PATCH macro has two different kinds of operands: task-oriented and work-oriented.

Task-oriented operands are used only at task creation and, if the task already exists, they are ignored (priority, queue length, target partition).

Work-oriented operands are relevant with every execution of the PATCH macro (entry point name, queue position, ECB address, FREE request).

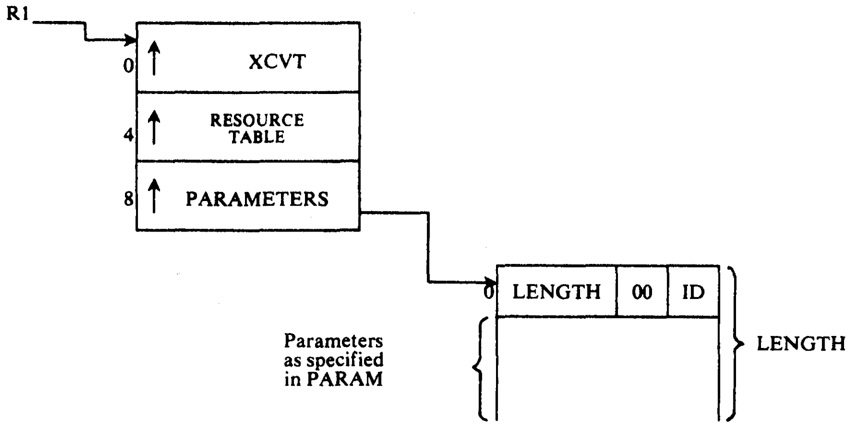
The various operands available can be used to control overall system overhead, core usage, task synchronization, and execution times. Their use should be considered carefully so that they correspond to the requirements of the task they affect.

Each time a program is called or executed as a result of a PATCH, a parameter list is passed to the program. These parameters may be used to identify the PATCHing program, the reason for the PATCH, to pass data or an address of data arrays, or, in general, to provide the PATCHed program any information it might need to execute a given This parameter list is always headed by one word containing the length of the parameter area and the ID of the PATCH. The remainder of the list can be any combination of values and/or addresses needed by the PATCHed program.

When the PATCH SVC returns to the caller, register 15 will contain a return code. In addition, if the return code is less than or equal to 8 and was for an independent task, register 1 will contain the address of the TCB extension (TCBX). If this address is supplied with the next PATCH to the same task (TCBX=), it will speed up execution of the PATCH SVC routine.

When the "called" program gains control as the result of a PATCH, register 1 will contain the address of a 3-word block which contains the address of: (a) XCVT, (b) resource table, (c) the parameters specified in the PATCH macro. The address of XCVT will be used as input to many Special Real Time Operating System macros as the keyword operand DCVTR or DCVTLOC. The resource table is a doubleword which is allocated and set to zero when the task is created and is maintained as a task resource as long as the task is in existence. The user may store data into the resource table and have the data preserved between independent task (program) executions eventhough the program may be deleted or a different program may be executed under the same task.

PATCH Input Parameter Format



```
[symbol]   PATCH   ┌ TASK = name                  ┐
                   │                  ┌ (r)     ┐ │
                   │ TASKLOC =        └ address ┘ │
                   └                              ┘
                   ┌ EP =    (name[,DELETE])       ┐
                   └ EPLOC = (address[,DELETE])    ┘
                   ┌        ┌ (r)    ┐ ┐
                   │ ,QL =  └ number ┘ │
                   └                   ┘
                   ┌          ┌ FIRST  ┐ ┐
                   │ ,QPOS =  │ LAST   │ │
                   └          └ DPATCH ┘ ┘
                   ┌                        ┌ (r)   ┐  ┐
                   │ PRTY = ( taskname,     └ value ┘) │
                   │           ┌ (r)     ┐             │
                   │ PRTYLOC =(└ address ┘,value)      │
                   └                                   ┘
                   ┌         ┌ (r)     ┐               ┐
                   │ ,ECB =( └ address ┘  [,REPATCH] ) │
                   └                                   ┘
                   ┌         ┌ ┌ AT  ┐ ┌           ┐ ┐ ┐
                   │         │((│ AP  │ │ ,   (r)   │)│ │
                   │ ,FREE = │ └ len ┘ └ ,address  ┘ │ │
                   └         └  p                   ┘ ┘
                   ┌          ┌ (r)     ┐ ┐
                   │ ,TCBX =  └ address ┘ │
                   └                      ┘
                   ┌         ┌ OWN    ┐ ┐
                   │ PTN =   │ MASTER │ │
                   │         │ SLAVE  │ │
                   └         └ FIND   ┘ ┘
                   ┌          ┌ L                      ┐ ┐
                   │ ,SUPL =  │ (E,   ┌ address ┐)     │ │
                   └          └       └   (r)   ┘      ┘ ┘
                   ┌        ┌ (r)   ┐ ┐
                   │ ,ID =  └ value ┘ │
                   └                  ┘
                   ┌            (r)   ┌ (r),...,(rn) ┐  ┐
                   │ ,PARAM = ( p1,   └ p2,...,pn    ┘) │
                   └                                    ┘
                   ┌           ┌ L                     ┐ ┐
                   │ ,PROBL =  │ (E,   ┌ address ┐)    │ │
                   └           └       └   (r)   ┘     ┘ ┘
                   ┌ ┌ ,DCVTR =  r                   ┐ ┐
                   │ │                ┌ (r)     ┐    │ │
                   └ └ ,DCVTLOC =     └ address ┘    ┘ ┘
Where 'r' is a general purpose register, 2-12.
```

TASK=name

Specifies a 1 to 8 character name which is the name of the task or queue holder being referenced by this PATCH. If the task does not exist, one by that name will be created.

TASKLOC=address

Specifies the address of a 1 to 8 character task name. The name must be on a fullword boundary, be left-justified and padded on the right with blanks, if necessary, to complete eight characters. The address can be any format valid with an RX-type instruction.

EP=

Specifies a 1 to 8 character valid program name which is the name of the program to be scheduled under the task being created with the PATCH. If DELETE is specified and this is the only task using the program, a DELETE is issued for the EP name after processing for this PATCH completes. DELETE may be abbreviated as DEL. If EP is not specified and an ID other than 255 is specified, the PATCH will fail during an attempt to LOAD a program with a name of blanks.

EPLOC=address

Specifies the address of a 1 to 8 byte program name. The program name must begin on a fullword boundary, be left-justified and padded on the right with blanks, if necessary, to complete eight characters. The address can be in any format valid with an RX-type instruction. DELETE has the same meaning as with EP=.

QL=

Specifies the limit number of WQEs that may be queued to the independent task in addition to the one that might be currently executing. This parameter is meaningful only for new, independent tasks and is ignored otherwise. Any decimal value from 0 to 255 may be specified; the default value is 1. If (r) is specified, the value is assumed to be in the low-order byte of register r.

If zero is specified as the queue length, the task accepts one PATCH, works on it and, when completed, waits for the next request. If a PATCH is issued for that task while the task is busy, it is not executed.

If the queue length is 1, the task can accept one PATCH even while it is busy.

When a task completes processing the current request, the top element on the queue will be executed next.

QPOS=

Specifies where in the task work queue this work request is to go if the task is currently busy. FIRST indicates that it is to be placed so as to be processed before those already in the queue. LAST indicates that those already in the queue should be processed before this request. If DPATCH is specified, the processing for this PATCH will not be executed until a DPATCH is issued for this task. Only one PATCH with QPOS=DPATCH is allowed to each task. QPOS=DPATCH is not allowed if TASK= specifies a queue holder.

PRTY=

Specifies a task name and a value which will determine the priority of the new task. The value (0-255) will be subtracted from the dispatching priority of the specified task. If (r) is specified, the value is assumed to be in the low-order byte of register r. If a value is omitted, zero is assumed.

PRTYLOC=

Has the same function as PRTY except that the task name is an 8-byte field at the address specified. The name must be left-justified and padded on the right with blanks. The value is specified exactly as with the PRTY=operand.

ID=

Specifies a decimal value from 0-254 to be passed as a parameter to the PATCHed task's program. If (r) is specified, the ID value should be in the low-order byte of the register. An ID of 255 is special. A PATCH request with an ID of 255 will cause a task to be created and initialized if it does not already exist and the module to be loaded. It will work its way to the top of the queue, but the program will never be entered. This provides a task pre-initialization capability. If ID is omitted, a default value of 0 is assumed. If the ID is 255 and EP is not specified, the task will be created and no program will be loaded.

PARAM=

Specifies one or more parameters which will be passed to the PATCHed task's program. The parameters may be any values or addresses meaningful to the program. If (r) is specified for a parameter, the value must be contained in the register r; otherwise, the parameter expands as an A-type address constant. Note that if only one parameter is specified and it is in a register, two sets of parentheses are required.

ECB=

Specifies the address of an ECB which may be used in a WAIT macro. This ECB is posted when processing for this PATCH completes or when the work represented by the PATCH is purged. The REPATCH option causes the ECB to be posted with the address of the REPATCH list (REPL) to be used in the REPATCH macro if this PATCH is not serviced because of a QPOS=FIRST PATCH with the queue being full. If REPATCH option is specified and the REPATCH occurs (ECB posted with a completion code of X'44'), a REPATCH macro must be issued. See description of the REPATCH macro. Note that if only ECB address is specified and it is in a register, two sets of parentheses are required.

FREE=

Specifies that a work space is to be passed to the PATCHed task and is to be freed after the work queue element built in response to this PATCH is completed (either GETMAINed area or GETWA area) or after the PATCHed task has terminated (GETWA storage areas only). The area must have been obtained via a GETMAIN from subpool zero or a GETWA and must be part of the partition where the work represented by this PATCH is to be executed. If REPATCH option is specified and the ECB is posted, the area will not be freed immediately. However, the area will be freed as a result of the mandatory REPATCH macro. It is invalid to code this operand if the PATCH goes to the other partition and the REPATCH option is specified.

A work area originally obtained via a GETMAIN macro call may be freed by specifying either the length and address of the work area or FREE=P. If FREE=P is specified, the problem program parameters (ID and PARAM or PROBL=) are FREEMAINed. A GETMAINed area will be FREEMAINed after the execution of the work represented by this PATCH.

A work area originally obtained via a GETWA macro call may be freed by specifying either AP or AT and the address of the work area. An AP request will cause the storage to be FREEWAed after the execution of the work represented by this PATCH. An AT request will cause the storage to be freed whenever the PATCHed task is terminated.

Any storage area associated work queue built in response to a successful PATCH (return code less than eight) and later removed from the PATCHed tasks' work queue chain before it can be executed, will be freed when the work queue is purged.

Any format valid for an RX-type instruction can be specified for the length or address.

TCBX=

Specifies the address of the TCB Extension Control Block (TCBX) for an existing independent task. If TCBX is specified as a register, TCBX=(r), that register is assumed to contain the actual TCBX address. If TCBX is specified as a relocatable expression, TXCB=addr, the TCBX address will be loaded from the specified address. The TCBX address is returned in register 1 after each successful PATCH to an independent task. Use of this operand with all PATCHes to the same task after the initial PATCH will reduce system processing time. Note that other parameters must still be specified for verification or in the event the task has been DPATCHed.

PTN=

In two-partition operation, this operand defines the target partition for the PATCH. OWN means that the target partition is the partition that executes the PATCH; MASTER defines the MASTER defines the master partition as the target partition. SLAVE defines the slave partition as the target partition; if SLAVE is coded and two-partition operation is not initialized (no MASTER/SLAVE control cards in the SYSINIT input stream), the PATCH will be rejected, and a return code is passed back in register 15. FIND causes the SVC to search fcr the specified task in the patchor's own partition; if it is found, it is used and the search exits. If it is not found, a switch is made to the other partition which is searched also. If a task by the specified name is not found in either partition, the SVC routine switches back to the patchor's own partition and behaves as if OWN was coded.

If the PTN operand is not coded, it defaults to OWN.

If two-position operation is not specified at the Special Real Time Operating System SYSGEN, this parameter is ignored.

SUPL=

Specifies a list or execute form of the PATCH supervisor operands. If the list form, SUPL=L is specified, no executable code is generated; therefore, all register notations are ignored as well as TASKLOC, EPLOC and PRTYLOC. With the execute form, SUPL=(E,addr), the address specifies a SUPL=L form parameter list, and any additional operands specified cause executable code to be generated which modifies the remote parameter list before the SVC instruction is generated. The address can be in any format valid with an RX-type instruction. SUPL=L and PROBL=L cannot both be specified.

PROBL=

Specifies a list or execute form of the problem parameter operands, ID and PARAM. PROBL=L generates a parameter list, and PROBL=(E,address) specifies the address of such a list in an execute form of the PATCH macro. Both PROBL=L and SUPL=L cannot be specified. Note: If the length of the Note: If the length of th PROBL parameter is equal to or less than eight bytes, the entire parameter is moved into a supervisor area. This will allow the use of a single PROBL list even though the ID might vary with each individual PATCH execution.

DCVTR=r

Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses having the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

When control is returned, register 15 contains one of the following return codes:

Decimal Code	PATCH Executed	Description
2	YES	TCBX=Address specifies the address of a TCBX which does not have the same name as specified in the SUPL. The proper TCBX is found or a new task is created.
4	YES	QPOS=FIRST caused loss of previous WQ.
6	NO	Specified task is in the no-PATCH state.
8	NO	Queue full.
10	NO	PRTY task name does not exist.
12	NO	Invalid PROBL parm list or address.
14	NO	Invalid SUPL parm list or address.
16	NO	DPATCH queue overflow.
18	NO	Invalid FREE=operand.
20	NO	DPATCH in progress for this task.
22	NO	PTN=SLAVE requested but not initialized.
28	NO	No CBGET storage for TCBX, WQE or LCB.
30	NO	Task name specified is a queue processor.
32	NO	Task name specified is a queue processor and QPOS=DPATCH.

ECB COMPLETION CODES:

High-Order Byte Code	Low-Order 3-Byte Code	Description
X'40'	Same as register contents from program	Successful completion.
X'42'	Zero	DPATCH occurred before work could be executed.
X'44'	Address of block or zero	REPATCH A PATCH with QPOS=FIRST forced this PATCH out of queue.
X'48'	Zero	BLDL failed (member not found or I/O error).

X'4C'	ABEND code	Task abnormally terminated while processing this request.
X'4F'	Same as register 15	The PATCH parameters were specified as part of a PTIME macro. The PTIME specification has been deleted, and no more PATCHes will be executed.

Relationship of Patch Operands to type of Task

Operands or Suboperand	Create Independent Task	Queue Independent Task	Dependent Task	Program Input Parameters
TASK/ TASKLOC	R	R	I	
EP/ EPLOC	R	R	R	
DELETE*	O	O		
PRTY/ PRTYLOC	O	N	O	
QL	O	N	N	
QPOS	O	O	O	
DPATCH*	O	O	O	
ECB	O	O	O	
REPATCH*	O	O	O	
FREE	O	O	O	
P*	O	O	O	
TCBX		O		
ID				O
PARAM				O

R = Required
O = Optional
N = Ignored
I = Invalid

*Suboperand of Preceding Operand

Figure 2-34.

PTIME

The PTIME macro provides Special Real Time Operating System time management services to the user. The macro causes a task to be PATCHed at a specified or relative time. Optionally, this PATCH can be repeated at a specified cycle interval continuously or for a certain number of PATCHes. The PTIME macro also allows previous PTIME calls to be modified or deleted. An additional function of the PTIME macro allows access to the correct Special Real Time Operating System time and date.

The following operands are available for the PTIME macro.

```
[symbol]     PTIME     ( ADD )
                       ) MOD (
                       ) DEL (
                       ( RET )

                       [          ( (REL)  (,A=address           ) ) ]
                       [ ,START=  ( )TOD(  ),   (r)              ( ) ]
                       [          ( (ADJ)  (, nH ,nM ,n .n S     ) ) ]

                       [ (,STOP=    (same format as START) ) ]
                       [ )                                 ( ]
                       [ ),COUNT=  (  (r)   )              ( ]
                       [ (         (number  )              ) ]

                       [            ( ( A=address      ) ) ]
                       [ ,INTRVAL=  ( )     (r)        ( ) ]
                       [            ( ( nH ,nM ,n .n S ) ) ]

                       [          ( NO ) ]
                       [          ) U  ( ]
                       [ ,PURGE=  ) C  ( ]
                       [          ( W  ) ]

                       [,PTID= (value)]
                       [       ( (r) )]

                       [ ,MF= (     L            ) ]
                       [      )(E,  (   (r)    ) ( ]
                       [      (     (address   ) ) ]

                       [ (,DCVTR=   r            ) ]
                       [ )                       ( ]
                       [ ),DCVTLOC= (address)    ( ]
                       [ (          (  (r)  )    ) ]

                         [PATCH operands  (See PATCH Macro)]

where 'r' is a general purpose register 2-12.
```

All time values are specified in the same format. The time is specified explicitly by hours, minutes, seconds, or any combination of the three as long as one is specified. The time value must not exceed 24 hours.

Examples are as follows:

3 hours: 3H or 180M or 10800S
1 hour, 3 minutes, 1-1/2 seconds: 1H, 3M, 1.5S or 3781.5S.

If (r) is specified, the time in hundredths of seconds is in register r. The A= suboperand allows the time value to be specified in a fullword at the address specified. The time value in the word must be specified in hundredth of seconds. The address may be any RX-type address.

ADD
MOD
DEL

Specifies the type of PTIME service requested. If omitted or if ADD is specified, a PTIME queue element (PTQE) is activated which controls the PATCHes issued according to the PTIME request. Since the PTQE exists independently of the creating task and may be modified or deleted, the PTQE is referred to by the task name, entry point name, and ID value of the parameters referred to by the operands TASK=, TASKLOC=, EP=, EPLOC=, and ID= as defined in the PATCH macro. Either task name or entry point name must be specified with a modify (MOD) or delete (DEL), but the remaining two are optional. However, if only a task name or entry point name is specified, all PTQEs with that name are deleted or modified regardless of entry point names or ID values.

RET

Causes the system to return the current time in 10 millisecond units in register 0 and the address of the Special Real Time Operating System time array in register 1. This time is a Special Real Time Operating System time which can be synchronized with an external time source. The time and date are maintained in several formats and are updated periodically. Thus one PTIME RET call gives a routine the current time as long as the address of the array is retained. See the TIMED DSECT for a description of time formats. All other operands are ignored with RET.

START=

Specifies the time of the first PATCH to be executed. The first suboperand determines the meaning of the time value specified in the remainder of the operand. If REL is specified, or if the operand is omitted, the time of the first PATCH equals the current Special Real Time Operating System time plus the time value in the remainder of the operand. If TOD is specified, the first PATCH occurs when the Special Real Time Operating System time equals the time of day specified by the remainder of the operand. If this time is less than the current Special Real Time Operating System time, the first PATCH does not occur until the next day. If ADJ is specified, the time of the first PATCH is calculated by assuming the time value in the operand to be a TOD value, except that the time value in the INTERVAL= operand is repeatedly added to the assumed TOD or ADJ is specified and the calculated STOPTIME is less than the calculated START time until that value is greater than current Special Real Time Operating System time. This prevents the possibility of unintentionally specifying a TOD less than the current Special Real Time Operating System time and the first PATCH not occurring for almost 24 hours. Also this allows distribution of the time management processing by offsetting ADJ time relative to a standard time.

STOP=
Specifies the Special Real Time Operating System system time after which no more PATCHes are issued. If REL is specified, or if the operand is omitted, the stop time is equal to the current Special Real Time Operating System time plus the time value in the remainder of the operand. If TOD'is specified, the stop time is equal to the time value in the remainder of the operand. If ADJ is specified, the stop time is calculated by assuming the time value in the operand to be a TOD value except that the interval time is repeatedly added to the assumed TOD time until that value is greater than the current Special Real Time Operating System time. When either REL, TOD, or ADJ is specified and the stop time is less than the calculated start time, a 24-hour value is added to the stop time until the STOP time equals or exceeds the START time.

COUNT=
Is equal to the number of PATCHes that are issued before the PTIME control block is deleted. This operand is an alternative to STOP=. The count value can be specified in a register, but must not exceed a halfword value.

Note: If both SOPT= and COUNT= operands are specified, the COUNT field will be ignored. If neither operand is specified, the PTIME is assumed to be infinite, and PATCHes will be issued until a PTIME DEL or MOD is issued for that task and/or entry point name.

INTRVAL=
Is the interval between successive PATCHes. If this operand is omitted or less than the SYSGENed time interval, the SYSGENed time interval will be substituted. The time may be specified with the A= suboperand as described above.

PURGE=
Provides a method of deleting the task associated with a PTIME. This operand can be specified when the PTQE is created (i.e., with ADD or MOD) or when the PTQE is deleted (DEL). If PURGE= (U,C,or W) is specified, a DPATCH is issued when the PTQE is deleted. The operand U, C, or W, specifies the type of DPATCH to be issued (see DPATCH description). If the task is to be deleted when the PTQE is deleted automatically via the STOP or COUNT operand, the PURGE operand must be specified in an ADD or MOD PTIME for the PTQE. Specification of the PURGE operand with a DEL type overrides the operand specified when the PTQE was created.

PTID=
Is a four byte value used to uniquely identify a PTQE. If this operand is omitted on a PTIME ADD request, the Special Real Time Operating System will assign a PTID. The PTID is returned to the user in Register 1 on PTIME ADD or MOD requests. PTID of a fullword of zeros or blanks will be ignored. If register notation is used, the specified register must contain the ID to be used.

MF=
Are the list and execute forms of PTIME which are generated by specifying MF=L and MF=(E,address), respectively. The list and execute forms are not valid with the RET option since this form has no parameter list. Also, the PATCH operands cannot be specified with MF=L.

DCVTR=r
Where r is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)

Where r is the general purpose register (2-12) enclosed in parentheses having the address of a 4-byte memory location that contains the address of the XCVT.

DCVTLOC=address
Where address is the label of a 4-byte memory location that contains the address of the XCVT.

PATCH Operands
Specifies the PATCH to be issued. Any valid combination of PATCH operands can be specified. Note that the PATCH supervisor and/or program parameter list can be expended with the list form of PATCH and then specified in execute form, i.e., SUPL=(E,addr), PROBL=(E,addr).

Note that there are some restrictions to the use of PATCH parameters with PTIME:

QPOS=
DPATCH cannot be specified. LAST will be substituted.

FREE=
Can be specified, but the FREEMAIN will not be executed until the PTIME queue element (PTQE) generated by this PTIME is deleted. If the PTQE is not repeating, this will be like a normal PATCH.

When control is returned, register 15 contains one of the following return codes:

Option Return Code	RET	ADD	MOD	DEL
0	Successful	Successful	Successful	Successful
4	NA	Interval time less than SYSGEN time interval--SYSGEN time interval substituted	Interval time less than SYSGEN time interval--SYSGEN time interval substituted	NA
8	NA	NA	PTQE not found	PTQE not found
12	NA	NA	TASK or EP name not specified	TASK or EP name not specified
16	NA	No CBGET area	NA	NA
20	NA	Duplicate PTQE (i.e., a PTQE already exists with the same PATCH Parameter and PTID value.)	NA	NA
24	NA	Invalid PATCH parameters (e.g., invalid ECB address)	Invalid PATCH parameters (e.g., invalid ECB address)	Invalid PATCH parameters (e.g., invalid ECB address)

When the return code is 8 or greater, the PTIME was not successful, and the existing PTIME specification will not be changed.

TIME ARRAY (DPPCTIMA):

```
+TIMED     DSECT
+***
+*         TIME ARRAY DSECT
+***
+TIMEHS        DC      F'0' TOD IN 10 MIL UNITS
+TIMETOD       DC      F'0' TOD IN 10 MIL UNITS-HHMMSSTH
+TIMEJDAY      DC      F'0' JULIAN DATE-OOYYDDDC
+TIMEMDAY      DC      F'0' DAY OF MONTH DATE-OMMDDYYC
+TIMEEBC       DC      CL10' '     EBCDIC DATE-DD/MMM/YY
```

PTIME INPUT PARAMETERS:

```
+PTIMEL DSECT
+***
+*             PTIME INPUT PARAMETERS
+*
+*                 REG1=ADDR OF SUPERVISOR LIST (IF REG 0 ZERO)
+*                 REG0=0 = RET OPTION
+*                     =4 = ADD OPTION
+*                     =8 = MOD OPTION
+*                     =12= DEL OPTION
+***
+PTIMSFLG      DC      XL1'0'      TIME OPTION FLAG
+PTIMSTRT      DC      AL3(0)      START TIME VALUE (OR ADDRESS)
+PTIMIFLG      DC      XL1'0'      PURGE OPTION FLAG
+PTIMINTL      DC      AL3(0)      INTERVAL TIME VALUE (OR ADDRESS)
+PTIMEFLG      DC      XL1'0'      TIME OPTION FLAG
+PTIMSTOP      DC      AL3(0)      STOP TIME VALUE (OR ADDRESS)
+              ORG     PTIMSTOP
+PTIMCNT       DC      AL3(0)      COUNT VALUE
+PTIMPTCH      DC      A(0)        PATCH SUPERVISOR LIST
+PTIMPARM      DC      A(0)        PATCH PROBLEM LIST
+PTIMPTQE      DC      A(0)        PTQE ADDRESS
+PTIMLNGH      EQU     *-PTIMEL
+*                     PURGE OPTION FLAGS
+PTIMFPRG      EQU     X'01'       PURGE DPATCH=U
+PTIMFDPC      EQU     X'02'       PURGE DPATCH=C
+PTIMFDPW      EQU     X'04'       PURGE DPATCH=W
+*                     TIME OPTION FLAGS
+PTIMCFG       EQU     X'08'       THIS FIELD CONTAINS COUNT VALUE
+PTIMREL       EQU     X'01'       RELATIVE TIME
+PTIMTOD       EQU     X'02'       TOD TIME
+PTIMADJ       EQU     X'04'       ADJUSTED TIME
+PTIMADDR      EQU     X'80'       THIS FIELD CONTAINS TIME ADDRESS
+PTIMLN        EQU     *-PTIMEL
```

VALID PTIME OPERAND COMBINATIONS:

Operand	Option			
	ADD	MOD	DEL	RET
START	O	O		
STOP	O	O		
COUNT	O	O		
INTRVAL	O	O		
PURGE	O	O		
MF	O	O		
DCVTR	O	O	O	O
DCVTLOC	O	O	O	O
PATCH operands	R	R	R	

070372

R = Required
O = Optional

PURGEWQ

The PURGEWQ macro is used to selectively purge work requests to a specified independent Special Real Time Operating System task or queue holder. The selected work requests will be removed from the active work queue (i.e., a chain of work requests that have been generated in response to PATCH macro calls but have not been executed yet) or from the DPATCH work queue (i.e., a work request generated in response to a PATCH OPOS=DPATCH... macro call). Other work requests for that task will not be purged and will be allowed to execute normally.

```
symbol   PURGEWQ   {SUPL = {(r) }                                             }
                   {       {addr}                                             }
                   {                                      [      {OWN    }]   }
                   {EP = {(r) } [,TASK={(r) }]            [,PTN= {FIND   }]   }
                   {     {addr} [      {name}]            [      {MASTER }]   }
                   {                                      [      {SLAVE  }]   }
                    ID = { (r)  }
                         {value }

                   [,FREE = (({ (r)  },{ (r)    }))]
                   [         ({length},{address }) ]

                   [,OPT = { NOPOST                 }]
                   [       { WAIT                   }]
                   [       { (POST,  { (r)     })   }]
                   [       {         {ECB addr })   }]

                   [,DCVTLOC = {(r) } ]
                   [           {addr} ]
                   [                  ]
                   [,DCVTR   =  (r)   ]

                   [,MF = { L              }]
                   [      { (E,  {(r) })   }]
                   [      {      {addr})   }]

   where 'r' is a general purpose register (2-12)
```

SUPL=
Specifies a list form of the PATCH supervisor operands. PURGEWQ uses this list to obtain the entry point name, task name, and partition reference. The SUPL= option allows the user to use the same SUPL for PATCH macro calls and the PURGEWQ macro call. If register form is specified, the register contains the address of the SUPL.

EP=
Specifies a 1 to 8 character valid program name which is the name of the program that is scheduled to be executed in response to a previous PATCH macro call. The entry point name is used in conjunction with the ID value to identify the work requests to be purged. If register form is specified, the register contains the address of an 8-character field which contains the program name. The name must be on a fullword boundary, be left-justified and padded on the right, if necessary, to complete eight characters.

TASK=
Specifies a 1 to 8 character name which is the name of a previously created independent task being referenced by this PURGEWQ. The task name identifies the task for which work requests are to be purged. If omitted, the current task is assumed to be the one for which work requests are to be purged. If register form is specified, the register

contains the address of an 8-character field which contains the task name. The name must be on a fullword boundary, be left-justified and padded on the right with blanks, if necessary, to complete eight characters.

PTN=
In two-partition operation, this operand specifies the target partition for this PURGEWQ. OWN means that the target partition is the partition that executes the PURGEWQ; MASTER defines the master partition as the target partition; SLAVE defines the slave partition as the target partition; FIND causes the PURGE WQ subroutine to search for the specified task in the partition that executes the PURGEWQ; if it is not found, the other partition is searched. OWN is the default option.

ID=
Specifies a decimal value from 0 to 254 to be used in conjunction with the entry point name to identify the work requests to be purged. This is, the ID that was passed as a parameter on a previous PATCH macro call. A PURGEWQ request with an ID of 255 will cause all work requests to the specified task with the specified entry point name to be purged regardless of the ID value specified on the originating PATCH macro. If ID is omitted, a default value of 0 is assumed. If register form is specified, the register must contain the ID value in the low-order byte.

FREE=
Specifies the length and address of a GETMAINed area that is to be FREEMAINed when the last specified work queue has been purged. Both the length and address are required and identify the area of storage to be freed. Either the length or address or both may be in registers.

OPT=
Specifies the PURGEWQ option used to determine if the user is to be notified when all specified work requests have been purged or, in the case when a work request to be purged is currently active, have been completed. NOPOST indicates that the specified work requests are to be scheduled for purging but control is to be returned to the user and no indication will be made when the work requests have actually been purged. WAIT indicates that the user is to be placed in a wait state until all specified work requests have been purged or, if a work request to be purged is currently active, until that work request has completed normal processing. POST indicates that the specified work requests are to be scheduled for purging but control is to be returned to the user. The user will be notified via a POST to the specified ECB whenever all specified work requests have been purged or, if a work request to be purged is currently active, when that work request has completed normal processing. If register form is specified on the OPT = POST parameter, the register contains the address of the ECB to be posted.

DCVTLOC=
Specifies the address of a 4-byte memory location that contains the address of the XCVT. If register form is specified, the register must contain the address of the location that contains the address of the XC7T.

DCVTR=
The register specified contains the address of the XCVT.

MF=
Specifies the list to execute form of the PURGEWQ which are generated by specifying MF=Land MF=(E, address) respectively. The list form is not valid with the SUPL parameter.

Note: The EP, TASK, and/or PTN parameters cannot be used with the SUPL parameter. Either the SUPL or the EP parameter must be specified.

After completion of a PURGEWQ macro call Register 15 will contain a return code and register 1 will contain a count of the number of work requests purged or zero (depending on the return code). This count of work requests purged does not include the current work request, if applicable, since it was not actually purged.

Register 15	Register 1	Description
0	No. of work requests purged	Successful completion
4	0	Task was dormant (no active work requests
8	No. of work requests purged	One of the work requests is the currently active work request
12	0	No work requests found to be purged
16	No. of work requests scheduled to be purged	OPT= (POST, ECB addr) specified but due to PATCH return code, we are unable to WAIT until all work requests have been purged before posting the user ECB
20	0	DPATCH in progress for this task
24	0	Invalid PTN= specified
28	0	Invalid input address or unable to locate specified task.
32	No. of work requests purged	Task was for the current task and OPT=WAIT was specified. This may cause an interlock situation. Therefore the WAIT request is ignored.

PUTARRAY

The PUTARRAY macro is used to move data into one or more VS resident arrays of the data base. The data in the entire array based on the length defined through the offline utility will be replaced. This macro is not valid for use with blocked arrays. Where incr is specified, it may be any value from 1 to 255.

[symbol]	PUTARRAY	NUMBER=number,DATA= {(r) | address}
		NAME=name,DATA= {(r) | address}
		NAMELST= ({(r) | address} [,incr])
		ADDRLST= ({(r) | address} [,incr])
		NUMBLST= ({(r) | address} [,incr])
		{,DATALST=({(r) | address} [,incr])}
		[{,DCVTR= r | ,DCVTLOC= {(r) | address}}]

The parameters NUMBER=, NAME=, NAMELST=, ADDRLST=, and NUMBLST are mutually exclusive; only one may be specified.

NAME=
Is an 8-character name of a single array into which data is to be moved.

NUMBER=
Is an array number of a single array into which data is to be moved.

DATA=
Is used with NAME= or NUMBER= operand. The address from which data is to be moved into the specified array.

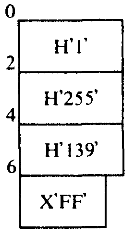
NAMELST=
Is the address of a list of 8-character array names for which data is to be moved. Incr is the value by which this address is to be incremented to locate the next name. If not specified, a value of 8 is assumed. The list must be terminated by a byte containing X'FF' in the position that would be occupied by the first byte of the next name.

ADDRLST=
Is the address of a list of data base array addresses as returned from a previous execution of the GETARRAY macro with NAME, NAMELST NUMBER, or NUMBLST specified and TYPE=ADDR. Incr is the value by which this address is to be incremented to locate the next array address. If incr is not specified, a value of 8 is assumed. If specified, must

be no less than 8. The list must be terminated by four bytes containing X'FFFFFFFF' in the position that would be occupied by the address of the next array. If the GETARRAY macro, TYPE=ADDR, and NAMELST or NUMBLST is used to build the list, it will place this flag at the end of the list.

NUMBLST=
Is the NUMBLST parameter that specifies the address of a list of 2-byte fields containing array numbers for which data is to be written. Incr is the value by which this address is to be incremented to locate the next number. If incr is omitted, a value of 2 is assumed. A value less than 2 must not be specified for incr. The list must be terminated by a byte containing X'FF' in the first byte of the 2-byte field which would be occupied by the next array number.

EXAMPLE: Number List



DATALST=
Is the address of a list of addresses into which the data from the specified array(s) is to be moved. The list must contain an entry for each array for which data is to be moved. This entry will contain a fullword address which identifies the memory address from which the first byte of the array data is to be moved. Incr is the value by which the address of the list is to be incremented to pick up the memory address to which the next array is to be moved. If incr is not coded, a value of 4 is assumed. If specified, must not be less than 4.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses that has the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

After execution of the PUTARRAY request, the return code in register 15 is set to zero to indicate successful completion or to four to indicate that the request could not be satisfied. This may be because of one or more of the following reasons:

- One or more of the named arrays is not defined to the system.
- A numbered array was requested which is higher than the highest numbered array defined to the system.
- A TYPE=DATA request was made for a direct access resident array.

PUTBLOCK

The PUTBLOCK macro will retrieve the data from user-allocated storage and place that data into blocked arrays. The macro may be used to write one or more blocks of data into one or more arrays. The arrays may be either virtual storage or direct access resident.

[symbol]	PUTBLOCK	$\left\{\begin{array}{l}\text{NAME= }\left\{\begin{array}{c}\text{name}\\\text{(r)}\end{array}\right\}\\\text{NUMBER= }\left\{\begin{array}{c}\text{number}\\\text{(r)}\end{array}\right\}\\\text{NAMELST= }\left\{\begin{array}{c}\text{address}\\\text{(r)}\end{array}\right\}\\\text{NUMBLST= }\left\{\begin{array}{c}\text{address}\\\text{(r)}\end{array}\right\}\end{array}\right\}$ $\text{,DATALST= }\left\{\begin{array}{c}\text{address}\\\text{(r)}\end{array}\right\}$ $\left[\left\{\begin{array}{l}\text{,DCVTR= r}\\\text{,DCVTLOC= }\left\{\begin{array}{c}\text{address}\\\text{(r)}\end{array}\right\}\end{array}\right\}\right]$

The parameters NAME, NUMBER, NAMELST and NUMBLST are mutually exclusive. The macro will not expand if more than one of these parameters is specified or if all of these parameters are omitted.

DCVTR=
Specifies a register (2-12) which contains the address of the XCVT.

DCVTLOC=
Specifies the address or a register (2-12) enclosed in parentheses which contains the address of a location which contains the address of the XCVT.

NAME=
Specifies the name or a register containing the address of the name of a named array into which data is to be written.

NUMBER=
Specifies the number or a register containing the number assigned to a numbered array into which data is to be written.

NAMELST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of array names into which data blocks are to be written. The name list will be a table of 8-byte entries with one valid array name in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the name list.

EXAMPLE: Name List

0	ARRAYNAM
8	HOUSTONƀ
16	TEXASƀƀƀ
24	X'FF'

NUMBLST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of array numbers into which data blocks are to be written. The number list will be a table of halfword entries with one valid array number in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the number list.

0	H'1'
2	H'255'
4	H'139'
6	X'FF'

DATALST=
Specifies the address or a register (not register 1) which contains the address of a user-constructed list of block numbers and of address from which the data blocks are to be moved. The data list will be a table of 6-byte entries. Each entry will contain a 1-byte flag field, a 3-byte area address and a 2-byte block number.

DATA LIST ENTRY DESCRIPTION:

0	1	2	3	4	5
FLAG BYTE	AREA ADDRESS			BLOCK NUMBER	

FLAG BYTE

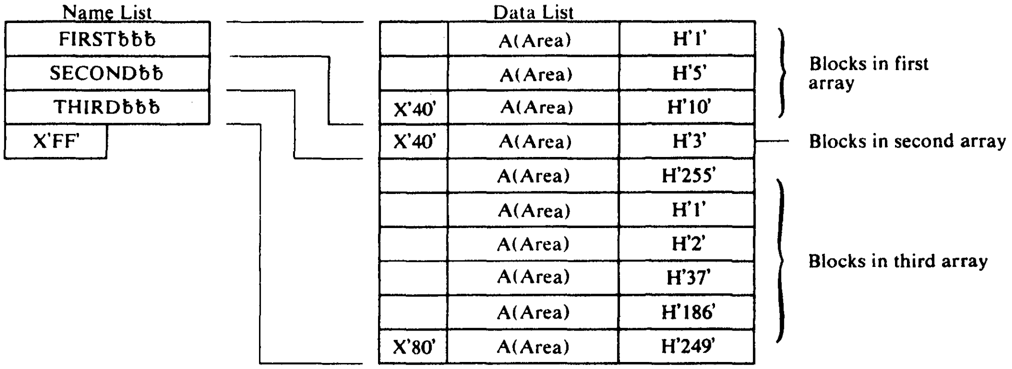
X'40' -- Indicates the last entry to be processed for a particular entry in the name list or number list.

X'80' -- Indicates the last entry in the data list.

AREA ADDRESS -- The address of a user-allocated area of storage from which the data block is to be moved. The area must contain the entry data block to be placed in the block.

BLOCK NUMBER -- The number assigned to the data block to be retrieved and placed in the array described in the Name List or Number List.

EXAMPLE: Data List and Name List



Name List

FIRSTbbb
SECONDbb
THIRDbbb
X'FF'

Data List

	A(Area)	H'1'	Blocks in first array
	A(Area)	H'5'	
X'40'	A(Area)	H'10'	
X'40'	A(Area)	H'3'	Blocks in second array
	A(Area)	H'255'	Blocks in third array
	A(Area)	H'1'	
	A(Area)	H'2'	
	A(Area)	H'37'	
	A(Area)	H'186'	
X'80'	A(Area)	H'249'	

Note: A zero returned in register 15 indicates successful completion. A non-zero returned in register 15 indicates that one or more errors were encountered during processing of this PUTBLOCK request. The high-order byte in register 15 contains a count of the number of errors encountered and the low-order three bytes contain the address of the first invalid array name or number.

PUTITEM

The PUTITEM macro is used to store data into one or more items of the data base. If another user of the data base is executing a data base access macro with PROTECT=YES, the operation of the PUTITEM macro will be delayed until all other users of the data base which have specified PROTECT=YES complete. This macro is not valid for use with direct access resident arrays. Where incr is specified, it may be any value from 1 to 255.

[symbol]	PUTITEM	$\begin{Bmatrix} \text{NAME= name} \\ \text{NAMELST= } \left(\begin{Bmatrix}\text{(r)} \\ \text{address}\end{Bmatrix} \text{ [, incr]}\right) \\ \text{ADDRLST= } \left(\begin{Bmatrix}\text{(r)} \\ \text{address}\end{Bmatrix} \text{ [, incr]}\right) \end{Bmatrix}$ $\left[\begin{Bmatrix} \text{,DCVTR= r} \\ \text{,DCVTLOC= } \begin{Bmatrix}\text{(r)} \\ \text{address}\end{Bmatrix} \end{Bmatrix}\right]$ $\left[\text{,BLKNO= } \begin{Bmatrix}\underline{\text{U}} \\ \text{number}\end{Bmatrix}\right]$ $\text{,DATA= } \left(\begin{Bmatrix}\text{(r)} \\ \text{address}\end{Bmatrix} \text{ [, incr]}\right)$

The parameters NAME=, NAMELST=, and ADDRLST are mutually exclusive; only one may be specified.

NAME=
Is an 8-character name of a single item for which data is to be stored.

NAMELST=
Is the address of a list of 8-character ITEM names for which data is to be moved. Incr is the value by which this address is to be incremented to locate the next name. If not specified, a value of 8 is assumed. If specified, the value must not be less than 8. The end of the list must be indicated by a byte containing X'FF' in the position that would be occupied by the first byte of the next name.

If the items are contained in blocked arrays, the block number for which data is to be retrieved must be specified in the halfword immediately following the 8-byte name. Also, the BLKNO=parameter should be specified and the incr must be coded as at least 10.

ADDRLST=
Is the address of a list of data base item addresses as returned from a previous execution of the GETITEM macro with NAME= or NAMELIST= specified and TYPE=ADDR. Incr is the value by which this address is to be incremented to locate the next item address. If incr is not specified, a value of 4 is assumed. The end of the list must be indicated by a 4-byte field containing X'FFFFFFFF' in the position that would be occupied by the next address. If the GETITEM macro with NAMELST option is used to build this list, it will place that value at the end of the list.

DATA=
Is the address from which the first data is to be moved. Data will be moved to the first ITEM specified, according to the length defined for that ITEM in the data base. Incr is the value by which the data address is to be incremented to determine the address to pick up the next data. If incr is not coded, the length of the item's is used.

BLKNO= __U__
       number

If U is specified or if the parameter is omitted, the array is unblocked. A number is used to specify that the data is to be retrieved from a blocked array(s). If NAME= was specified, number is the block number from which data is to be retrieved. If NAMELST= is specified, any number from 1 to 32767 may be coded to indicate that the block numbers are coded as part of the NAMELIST=.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=r
Where 'r' is the general purpose register (2-12) enclosed in parentheses that has the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

When control is returned, register 15 contains one of the following return codes:

Decimal Code	Description
0	Successful execution.
4	One or more of the item names specified could not be resolved or data was requested to be moved for the item with defined length of 0 bytes.
8	Invalid options were passed to the PUTITEM routine (probably the macro expansion had been modified).
12	A block number was specified for an unblocked array or a block number was specified that is greater than the highest block number defined for the array.
16	PUTITEM request for an item that is contained in a direct access array.

PUTLOG

The PUTLOG macro logs data base arrays on demand.

[symbol]	PUTLOG	{ NAME= {name | (r)}
		NUMBER= {number | (r)}
		NAMELST= {address | (r)}
		NUMBLST= {address | (r)} }
		[{ ,LOGHDR= {address | (r)}
		,BLKLIST=({address[,incr] | (r)}) }]
		[,PROTECT= {RISK | YES}]
		[{ ,DCVTR= r
		,DCVTLOC= {address | (r)} }]

The parameters NAME, NUMBER, NAMELST, AND NUMBLST are mutually exclusive. The macro will not expand if more than one of these parameters is specified or if all of these parameters are omitted.

DCVTR=
Specifies a register (2-12) which contains the address of the XCVT.

DCVTLOC=
Specifies the address or a register (2-12) enclosed in parentheses which contains the address of a memory core location which contains the address of the XCVT.

NAME=
Specifies the name or a register (2-12) which contains the address of a name of a named array from which data is to be logged.

NUMBER=
Specifies the number or a register (2-12) containing the number assigned to a numbered array from which data is to be logged.

NAMELST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of array names from which data is to be logged. The name list will be a table of 8-byte entries with one valid array name in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the name list.

EXAMPLE: Name List

Offset	Entry
0	ARRAYNAM
8	HOUSTONb̸
16	TEXASb̸b̸b̸
24	X'FF'

NUMBLST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of array numbers from which data is to be logged. The number list will be a table of halfword entries with one valid array number in each entry. The first byte past the last valid entry will be set to X'FF' to indicate the end of the number list.

EXAMPLE: Number List

Offset	Entry
0	H'1'
2	H'255'
4	H'139'
6	X'FF'

LOGHDR=
Specifies an address or a register containing the address of any array logging header. Information in this logging header will identify the copy of the array which is to be replaced in the log data set. The LOGHDR parameter cannot be specified if the BLKLIST parameter is specified.

The logging header is a 24-byte control block which precedes the array, both as the array exists in virtual storage and as it is written to the logging array. The logging header which was retrieved as part of a previous GETLOG macro may be used to replace that copy in the log data set.

BLKLIST=
Specifies the address or a register (2-12) which contains the address of a user-constructed list of block numbers and of core addresses from which data blocks are to be moved. The data list will be a table of 6-byte entries. Each entry will contain a 1-byte flag field, a 3-byte area address, and a 2-byte block number. This will allow the user to update selected segments of the DA log array for block VS resident arrays on demand basis. The latest log copy will be modified. However, the entire VS resident block is not necessarily logged; only the log block which contains the VS resident block specified will be updated. The actual log copy will not change when using this parameter; that is, repeated PUTLOG macro calls with BLKLIST parameters will update the same log copy.

BLKLIST ENTRY DESCRIPTION:

0	1	2	3	4	5
FLAG BYTE	AREA ADDRESS			BLOCK NUMBER	

FLAG BYTE

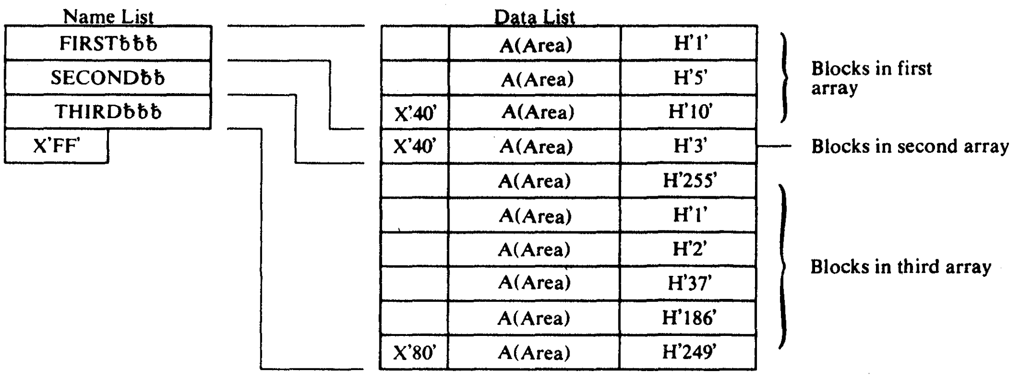
X'40' -- Indicates the last entry to be processed for a particular entry in the name list or number list.

X'80' -- Indicates the last entry in the data list.

AREA ADDRESS -- Not applicable for PUTLOG. The area is allocated so that list forms of PUTLOG and PUTBLOCK are the same.

BLOCK NUMBER -- The number assigned to the data block to be retrieved and placed in the array described in the name list or number list.

EXAMPLE: BLKLIST and Name List



Name List
FIRSTbbb
SECONDbb
THIRDbbb
X'FF'

Data List		
	A(Area)	H'1'
	A(Area)	H'5'
X'40'	A(Area)	H'10'
X'40'	A(Area)	H'3'
	A(Area)	H'255'
	A(Area)	H'1'
	A(Area)	H'2'
	A(Area)	H'37'
	A(Area)	H'186'
X'80'	A(Area)	H'249'

PROTECT=
If YES is specified, a lock will be set to prevent other programs that specify PROTECT=YES from accessing the data base while this PUTLOG is in the process of modifying the data base. If RISK is specified, the data will be moved without regard to other programs which may be accessing the data base.

Note: A zero returned in register 15 indicates successful completion. A non-zero returned in register 15 indicates that one or more errors were encountered during processing of this PUTLOG request. The high-order byte of register 15 contains a count of the number of errors encountered and the low-order three bytes contain the address of the first invalid array name or number.

RECORD

The RECORD macro is used to write data from programs in execution to a sequential data set. The data in the data set can then be retrieved at a later time through the playback function.

[symbol]	RECORD	ID= $\left\{\begin{matrix}\text{(r)}\\\text{number}\end{matrix}\right\}$, ADDR= $\left\{\begin{matrix}\text{(r)}\\\text{address}\end{matrix}\right\}$,COUNT= $\left\{\begin{matrix}\text{(r)}\\\text{number}\end{matrix}\right\}$ $\left[\left\{\begin{matrix}\text{,DCVTR= r}\\\text{,DCVTLOC=}\left\{\begin{matrix}\text{(r)}\\\text{address}\end{matrix}\right\}\end{matrix}\right\}\right]$

ID=
Is a unique 3-digit hex number (001-FFF) which identifies the data that is to be recorded (written) to a sequential data set. If (r) is specified, the register must contain the 3-digit hex number.

ADDR=
Is the address of the data that is to be recorded. If (r) is specified, the register must contain the address of the data to be recorded.

COUNT=
Is the number of bytes that is contained in the data. The maximum size is 65525 bytes. If (r) is coded, the specified register must contain the number of bytes to be recorded.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses that has the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

Code	Description
00	Normal Completion
04	ID is Disabled
12	End of data set reached or I/O error on output of data record. All data recording is disabled for this job step.

RETURN CODES. RECORD macro will issue return codes via register 15.

REPATCH

When a PATCH forces a WQE to fall out of the queue (QPOS=FIRST is specified and the queue was full), a Repatch List (REPL) will be constructed if the failing PATCH had REPATCH option specified. The user's ECB will be posted with a completion code of X'44' in the high-order byte and the address of the Repatch List in the three low-order bytes.

Note: The three low-order bytes represent a REPL address only if the REPATCH option was specified and the completion code (high-order byte) is X'44'.

<u>Warning</u>: If the REPATCH option was specified, and the ECB is posted with X'44', a REPATCH macro must be executed, so that the Repatch List built from Special Real Time Operating System Control Block Storage can be freed.

[symbol]	REPATCH	REPL= $\left\{\begin{matrix}\text{(r)}\\ \text{address}\end{matrix}\right\}$, TYPE= $\left\{\begin{matrix}\underline{\text{EXEC}}\\ \text{PURGE}\end{matrix}\right\}$ [,PATCH operand. . .] $\left[\left\{\begin{matrix}\text{,DCVTR= r}\\ \text{,DCVTLOC=}\left\{\begin{matrix}\text{(r)}\\ \text{address}\end{matrix}\right\}\end{matrix}\right\}\right]$

REPL=(r)
Where 'r' is the general purpose register (2-12) that contains the address of the REPL.

REPL=address
Where 'address' is the label of a 4-byte storage location that contains the address of the REPL, e.g., the label of the ECB that was posted with the address of the REPL.

TYPE=
Specifies whether the PATCH is to be retried (EXEC) or deleted (PURGE). Only one REPATCH is permitted for every original PATCH. If TYPE=EXEC is specified and the WQE is pushed out again, no REPL will be built. A TYPE=PURGE causes the FREE=, if specified in the original PATCH, to be issued and the Repatch List to be freed.

PATCH Operands

If any PATCH operands are specified with a REPATCH TYPE=EXEC, the REPATCH macro will internally invoke the PATCH macro and the code will be generated that modifies the REPL. Since the REPL supplied by the Special Real Time Operating System is in protected storage prior to issuing a REPATCH macro with PATCH operands, the user must obtain storage (length=REPLLNTH) and copy the supplied REPL (for that same length) into his own storage. Since one of the words in the Special Real Time Operating System supplied REPL contains its own address, the user's REPL will also have this address so that the REPATCH SVC routine can free its storage. Several restrictions for PATCH operands follow:

- SUPL, PROBL, ID, and PARAM must not be specified.
- If PRTY or PRTYLOC is specified, both subvalues (name, priority) must be specified.
- REPATCH option must not be specified.
- Care must be taken in modifying FREE=operands since the original FREE request has not been processed.

Specifying PATCH operands with a REPATCH TYPE=PURGE will not generate instructions to modify the Repatch List and will therefore have no effect on the execution of the REPATCH SVC routine.

DCVTR=r
Where 'r' is the general purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general purpose register (2-12) enclosed in parentheses having the address of a 4-byte location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte location that contains the address of the XCVT.

Note: The REPL DSECT can be obtained by the macro DPPXBLKS REPL=Y.

REPATCH RETURN CODES:
The REPATCH SVC routine returns a return code of 32 if an invalid TYPE or REPL address was specified. If the TYPE and REPL addresses are valid, the REPATCH SVC routine internally invokes the PATCH SVC routine and the return codes received upon return from PATCH are passed back to the user upon return from REPATCH.

CHAPTER 3. INSTALLATION GUIDE

INTRODUCTION

Three distinct phases must be considered prior to building a Special Real Time Operating System: pre-SYSGEN, SYSGEN, and system initialization. In addition, there are certain considerations prior to generating the host OS/VS1 system. These considerations and the building and running of the Special Real Time Operating System are discussed in the following sections.

The pre-SYSGEN phase of building the Special Real Time Operating System consists of performing such functions as copying libraries, creating SYSGEN input, and allocating data sets, i.e., the normal preparatory work that must be done for any SYSGEN.

The SYSGEN is the procedure by which the customer creates a Special Real Time Operating System tailored to his individual software requirements and hardware installation. SYSGEN comprises a series of OS/VS1 job steps for normal functions such as assemblies, link-edits, and copies.

System initialization is the process through which the Special Real Time Operating System is brought into virtual storage and initialized for a realtime run. When initialization is completed, the Special Real Time Operating System is operating.

In addition to building and running the Special Real Time Operating System, modifications may be made, to the data base for example, in an offline mode. To allow minor modifications without the necessity of a SYSGEN, an offline utility program is supplied with the Special Real Time Operating System. The use of this program is described in this section, as its primary function is the creation and modification of the customer's data sets.

OS/VS1 SYSGEN CONSIDERATIONS

The installation of the Special Real Time Operating System in an OS/VS1 system does not require any modifications to the VS system. However, certain OS/VS1 facilities must be provided to the Special Real Time Operating System through the OS/VS1 SYSGEN, and careful consideration should be given to other OS/VS1 SYSGEN options. In addition, the requirements of related PRPQs being installed along with the Special Real Time Operating System must be considered.

The Special Real Time Operating System requires three user-generated SVCs: a Type I, a Type II, and a Type IV. The SVC numbers may be any of the allowable OS/VS1 user SVCs. The SVCs should be generated disabled. An example of the generation of the Special Real Time Operating System SVCs during the OS/VS1 SYSGEN is shown below.

```
SPECSVCS        SVCTABLE        SVC-255-D1-S0,                *
                SVC-254-D2-S6,                                *
                SVC-253-D4-S6
```

OS/VS1 has reserved the names IEAXYZ1 through IEAXYZ5 for CSECTs that must reside in the V=R nucleus. If the installation requires that the Special Real Time Operating System be SYSGENed with the Computer Status Panel or an external time source, a CSECT named IEAXYZ5 will be generated. This precludes the use of this CSECT name by other programs that would reside in the nucleus.

Careful consideration should be given to multiple console support routing codes in the OS/VS1 SYSGEN, as they will affect the Special Real Time Operating System. (See MCS operand on VS SYSGEN macro.)

The allocation for the SYS1.MACLIB data set should be made for BLKSIZE=6080, if possible, to allow for conformity with the Special Real Time Operating System source data sets A5799AHE.SOURCE and A5799AHE.MSGFILE. If this size is not possible or practical, the Special Real Time Operating System data sets A5799AHE.SOURCE and A5799AHE.MSGFILE must be reblocked to the block size of the customer's SYS1.MACLIB data set. Also, the data sets named by the MACDSET=keyword and the ARRDSET=keyword must be allocated with the same block size as the SYS1.MACLIB data set.

PRE-SPECIAL REAL TIME OPERATING SYSTEM SYSGEN INITIALIZATION

Certain preparations must be made prior to the Special Real Time Operating System SYSGEN. Data sets must be allocated, modules moved or copied, and the Special Real Time Operating System distribution tapes must be restored to a direct access device.

The Special Real Time Operating System SYSGEN requires (as input) the OS/VS1 Stage 2 input stream in a sequential data set. This input must be saved when executing the OS/VS1 SYSGEN for this purpose.

The data sets required for the Special Real Time Operating System SYSGEN fall in to three categories, as shown in Figure 3-1.

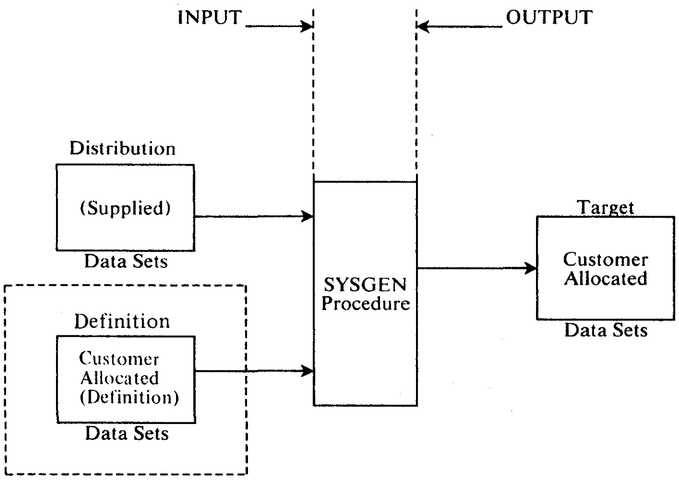


Figure 3-1. The Special Real Time Operating System SYSGEN Data Sets

The Special Real Time Operating System distribution data sets that are required are distributed to the customer on the tape sent from the IBM program library. The three distribution data sets are:

A5799AHE.SOURCE

A5799AHE.OBJECT

A5799AHE.MSGFILE

The definition data sets are optional and are not required for a Special Real Time Operating System only, or for a Special Real Time Operating

System and Display Management SYSGEN. However, if they are used, the customer must allocate them. The four definition data sets are configuration, software options, display, and data base. They will be named and allocated by the customer and each must be a partitioned data set.

The configuration and the software options data sets are required as input to Stage I of SYSGEN, only if the customer chooses to invoke the SYSGEN utility program DOMXSTG1 to do the SYSGEN. The alternative to invoking DOMXSTG1 is to code the Special Real Time Operating System SYSGEN macros in card image and to pass the cards to the OS/VS1 assembler, as is done for an OS/VS1 SYSGEN.

If DOMXSTG1 is invoked, however, the configuration definition and software options data sets must be created prior to Stage I. The The data sets must be partitioned card image and must contain the following:

- Configuration Data Definitions -- Each member must represent one System/7 or System/370 in the hardware configuration. All configuration data for each system of a computer hierarchy must be in this one data set. Each member name must be of the form S7xx or S370xx, where xx is the CPU identifier of that particular system. Macro statements in this data set must be those from the section: "Configuration Customer Definition Data Set Macros".

- Software Options Definition -- Each member must represent one System/7 or one System/370 in the configuration. All software options data for either system of a CPU hierarchy must be in this one data set. Each member name must be of the form S370xx or S7xx, where xx is the identifier of that particular system. Macros in this data set must be those from the section: "Software Customer Definition Data Set Macros".

The display and data base data sets are optional. When these data sets are used, they provide additional input to the offline utility program when it is invoked during Stage II of the Special Real Time Operating System SYSGEN. Their presence, which signifies additional processing in Stage II, is indicated by pointing to the data sets by the DISDSET and DBDSET keywords on the GENEMS macro. When used, the data sets must be partitioned card image, with a BLKSIZE equal to the customer's SYS1.MACLIB data set and must contain the following:

Data Base Definition -- Each member must contain at least one data base array definition that the customer desires to be placed in the final system by the SYSGEN process. All data base definitions for all System/370s and System/7s may be placed either in one data set or in separate data sets.

The output is placed in the target data sets by the SYSGEN procedures. SYSGEN is informed of these data sets through the GENEMS macro, as described in the SYSGEN macros section of this manual.

A number of OS/VS1 macros are required by the Special Real Time Operating System. These macros exist on the OS/VS1 distribution library SYS1.AMODGEN. Prior to the Special Real Time Operating System SYSGEN, the required macros must be moved from SYS1.AMODGEN to SYS1.MACLIB. Below is an example of the JCL needed to move the required macros. The members named in the SELECT statements are the equivalent of a list of members that must be in SYS1. If the member is already in SYS1.MACLIB, it will not be copied.

```
//COPY         JOB     ACCOUNT,PROGRAMMER
//             EXEC    PGM=IEBCOPY
//             DD      SYSOUT=A
//DDIN         DD      DSN=SYS1.AMODGEN,UNIT=2314,
//                     VOL=SER=TEST01,DISP=SHR
//DDOUT        DD      DSN=SYS1.MACLIB,DISP=OLD
//SYSIN        DD      *
    COPY OUTDD=DDOUT,INDD=((DDIN,R))
    SELECT MEMBER=(IEFTIOT1,IHBPSINR,IKJTCB,IHAFLC)
    SELECT MEMBER=(IEFUCBOB,IEFJFCBN,IHAPDDT,IHARB)
    SELECT MEMBER=(IEZDEB,IEZXRB,IHBRELNO,CVT,SYNCH)
```

The data shown underlined in the previous example will need to be changed to suit each customer's requirements.

The Special Real Time Operating System modules are distributed on tape from the program library; however, prior to the Special Real Time Operating System SYSGEN, the distributed data sets must be restored to a direct access device. A job stream is provided as a first file on the distribution tape that will move the libraries. To execute this job stream, the user must first place in his SYS1.PROCLIB a PROC named PPDSDEF. This procedure is the first job step encountered in the job stream in the distribution package. The only function of this PROC is to make DD statements available to succeeding job steps.

The following statements are required in this procedure:

```
//STEPABC      EXEC    PGM=IEFBR14
//DOMVOL       DD      UNIT=SYSDA,DISP=OLD,
//                     VOL=SER=PACK01
```

The step name must be STEPABC; the DD card must be named DOMVOL and must have the volume serial number of the pack to which the Special Real Time Operating System modules are to be moved.

After executing the described procedure as the first job step, subsequent job steps in the same job stream can determine the target pack for the Special Real Time Operating System modules by making a reference of the form:

```
VOL=REF=*.A.STEPABC.DOMVOL
```

The following is an example of the JCL and control cards required to add procedure PPDSDEF to the SYS1.PROCLIB.

```
//ADDPROC      JOB     ACCOUNT,'PROGRAMMER'
//             EXEC    PGM=IEBUPDTE,PARM='NEW'
//SYSUT2       DD      DSN=SYS1.PROCLIB,DISP=OLD
//SYSPRINT     DD      SYSOUT=A
//SYSIN        DD      DATA
./  ADD                NAME=PPDSDEF,LIST=ALL
./  NUMBER             NEW1=00,INCR=10
//STEPABC      EXEC    PGM=IEFBR14
//DOMVOL       DD      UNIT=SYSDA,DISP=OLD,VOL=SER=PACK01
./             ENDUP
/*
```

The data shown underlined in the preceding example will need to be changed to the customer's installation requirements.

With the member PPDSDEF in the SYS1.PROCLIB, the Special Real Time Operating System modules may be restored by entering the following start command on the OS/VS1 console.

```
START RDRT,181,LABEL=(1,NL)
```

Note: 181 should be replaced by the I/O device address where the distribution tape is mounted.

If related PRPQs or program products are being SYSGENed along with Special Real Time Operating System, the Special Real Time Operating System libraries should be restored first. If the Display Management PRPQ 5799-AFD is also being SYSGENed, it should be done next, followed by the restoration of other related products' tapes.

If supplementary material has been ordered by the customer, the tape containing this material can be restored to disk by executing the same start command.

The optional material is added to the existing distribution data sets. The basic material must be restored to disk before the optional material. When the optional material is restored, the disk data sets are already defined and cataloged. Consequently, the PROC PPDSDEF is not needed.

THE SPECIAL REAL TIME OPERATING SYSTEM DATA SET ALLOCATION

The user must, prior to Stage II of SYSGEN, allocate target data sets. The following example gives the recommended space allocation required for the Special Real Time Operating System SYSGEN:

OBJDSET	-	SPACE=(CYL,(1,1,50))
LMDSET	-	SPACE=(CYL,(1,1,50))
MACDSET	-	SPACE=(CYL,(1,1,50))
ARRDSET	-	SPACE=(CYL,(1,1,50))
DB1DSET	-	SPACE=(CYL,(2,,50))
DB2DSET	-	SPACE=(CYL,(2))
DB4DSET	-	SPACE=(CYL,(2,,50))
PLIDSET	-	SPACE=(CYL,(1,1,50))
PLSDSET	-	SPACE=(CYL,(1,1,50))
FORDSET	-	SPACE=(CYL,(1,1,50))

These figures can be used in conjunction with the chart in the description of the GENEMS macro to allocate the Special Real Time Operating System target data sets. The above space is for a 3330 direct access storage device and is for a Special Real Time Operating System only SYSGEN. The DB1DSET, DB2DSET, and DB4DSET data sets may have to have larger space allocation depending on the user's data base and messages. If these data sets are not going to be supported by duplicate data set support, secondary allocation may be requested.

FAILOVER/RESTART STORAGE REQUIREMENTS

Failover/Restart Write requires an amount of virtual address space determined by the following formula. In addition, the entire area is page fixed during Restart Write.

Address space required = 12,288 + (k*2048)

where $k = (T_L + 8 + 2047)/2048$
and from the above calculation for k,
k is the integer and any fractions
are ignored.

where T_L = Maximum blocksize of the device upon which
the Failover/Restart data set is allocated
(13030 for a 3330, 7294 for a 2314, etc.)

Example
for a 3330, k would be
k = (13030 + 8 + 2047)/2048
k = 7 ignoring the fraction;

therefore, the address space required
= 12,288 + (7*2048)
= 26,624 bytes.

The entire address space used is released when Restart Write is completed.

The amount of direct access space required for the Failover/Restart data set can be computed as follows:

Number of tracks required = 1 + A + B + C + D + E

where: A = Space required for the real storage portion of the data set

B = Space required for duplicating the active paging data set entries.

C = Space required for the SYS1.SYSJOBQE dump

D = Space required for the SWADS dump for the MASTER partition

E = Space required for the SWADS dump for the SLAVE partition.

In the following formulas the following terms and functions apply.

T_L = Maximum blocksize of the Failover/Restart set

N_P = Number of active paging entries on the SYS1.PAGE data set

D_P = Number of devices containing SYS1.PAGE data sets

R = Real storage size

TRUNC = A function that takes the integer portion of a quantity only and discards the fraction

N_{UQ1} = Number of 24-byte records in SYS1.SYSJOBQE

N_{UQ2} = Number of 176-byte records in SYS1.SYSJOBQE

N_{S1} = Number of records in SWADS for the MASTER partition

N_{S2} = Number of records in SWADS for the SLAVE partition

$$A = \frac{\text{TRUNC}(R/2048)}{\text{TRUNC}(T_L/2048} + 4$$

$$B = \frac{N_P}{\text{TRUNC}(T_L/2056)} + DP$$

$$C = \text{TRUNC}\left(\frac{N_{UQ1}}{\text{TRUNC}(T_L/32)}\right) + \text{TRUNC}\left(\frac{N_{UQ2}}{\text{TRUNC}(T_L/184}\right) + 2$$

D = 0 if SWA is used in MASTER partition; or

$$D = \text{TRUNC}\left(\frac{N_{S1}}{\text{TRUNC}(T_L/184)}\right) + 1$$

E = 0 if no SLAVE partition or SWA is used in SLAVE; or

$$E = \text{TRUNC}\left(\frac{N_{S2}}{\text{TRUNC}(TL/184)}\right) + 1$$

Estimation of quantity N/P requires consideration of the size of the link pack area, BLDL list, JES options, numbers of partitions, and size and current allocation of active partitions.

THE SPECIAL REAL TIME OPERATING SYSTEM SYSGEN

System generation (SYSGEN) is the procedure whereby the Special Real Time Operating System and associated PRPQs are combined to create a realtime system tailored to the needs of an individual user. The Special Real Time Operating System SYSGEN is analogous to the OS/VS1 SYSGEN procedure used to create an operational OS/VS1 system. The Special Real Time Operating System SYSGEN is normally performed only when major changes to the system occur. Data of a more changeable nature is entered into the system through the offline utility.

The Special Real Time Operating System SYSGEN process is patterned after the OS/VS1 SYSGEN procedure. It is comprised of two phases, Stage I and Stage II. Stage I creates the job stream input for Stage II. Stage I can be executed either by using the Special Real Time Operating System utility DOMXSTG1, or by directly invoking the OS/VS1 assembler or the assembler H program product (5734-AS1) to assemble the Stage I input cards.

The direct implementation of the assembler method can be used if the Special Real Time Operating System only is being SYSGENed, or if the Special Real Time Operating System and the Display Management PRPQ (5799-AFD) are being SYSGENed. For the Special Real Time Operating System only, the required SYSGEN macros are coded and passed to the assembler. For the Special Real Time Operating System and the Display Management PRPQ, the macros are also passed to the assembler; however, care must be taken to ensure that the SYSGEN macros are properly sequenced for the member. (CONFIGH, DEFDEV, and GENEMS is the correct order.)

If other related PRPQ or program products are being SYSGENed, the utility program DOMXSTG1 should be used.

If the customer has coded his SYSGEN macros and configuration macros and placed them in configuration and software option definition data sets, DOMXSTG1 may be used. This is shown in Figure 3-2.

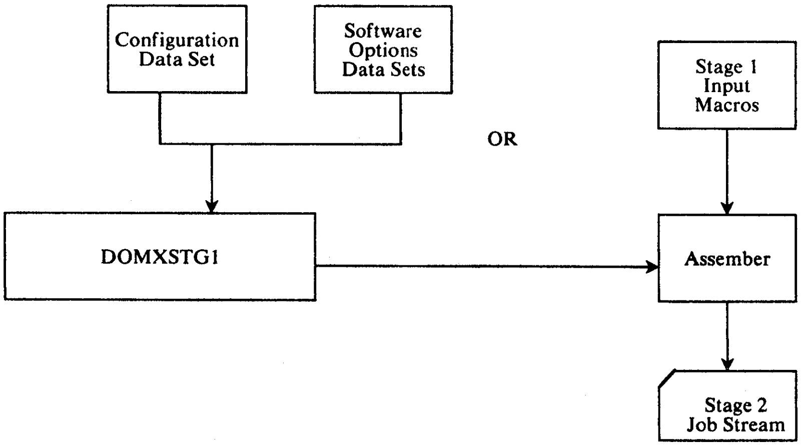


Figure 3-2. The Special Real Time Operating System - SYSGEN - Stage I

The following statement is the JCL required to invoke the DOMXSTG1 utility.

```
//STG1          JOB    ACCOUNT, PROGRAMMER
//              EXEC   PGM=DOMXSTG1,PARM='S37001'
//STEPLIB       DD     DSN=A5799AHE.OBJECT,DISP=SHR
//SYSLIB        DD     DSN=A5799AHE.SOURCE,DISP=SHR
//SOFTOPT       DD     DSN=(USER CREATED),DISP=SHR
//CONFG         DD     DSN=(USER CREATED),DISP=SHR
//SYSPRINT      DD     SYSOUT=A
//SYSUT1        DD     UNIT=SYSDA,SPACE=(CYL,5)
//SYSUT2*       DD     UNIT=SYSDA,SPACE=(CYL,5)
//SYSUT3*       DD     UNIT=SYSDA,SPACE=(CYL,5)
//SYSGO**       DD     UNIT=2400,DISP=(,PASS),LABEL=(,NL),
//                     DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSIN         DD     UNIT=SYSDA,SPACE=(CYL,(1,1))
//                     DCB=BLKSIZE=6080
```

 *Not required for assembler H
**If assembler H is used, SYSGO should be SYSLIN.

The JOB card should contain proper accounting information and any other data required in the customer's account. The PARM= field on the EXEC card identifies the system to be built. If the assembler H program product is to be used for the Stage I SYSGEN, the PARM field would be PARM='H,S37001'. The SOFTOPT and CONFG DD cards must define the software options and configuration definition data sets respectively, as these data sets are customer built. The SYSUT1, SYSUT2, and SYSUT3 DD cards may be coded to suit the customer's account. The The SYSGO (or SYSLIN for assembler H) DD cards specify the output data set, and if coded as shown in the previous example, Stage II of the Special Real Time Operating System SYSGEN can be started by starting an OS/VS1 reader to the data set, e.g., START RDRT,180,LABEL=(1,NL). DOMXSTG1 will place the source code macros from the configuration and software options data sets in the SYSIN data set and pass it as input to the assembler.

The output from the Stage I is an OS/VS1 job stream which, when executed, comprises the Stage II of SYSGEN. Stage II creates the Special Real Time Operating System from the input data sets, and places the components of the system in the target data sets pointed to by the GENEMS macro. This is shown in Figure 3-3.

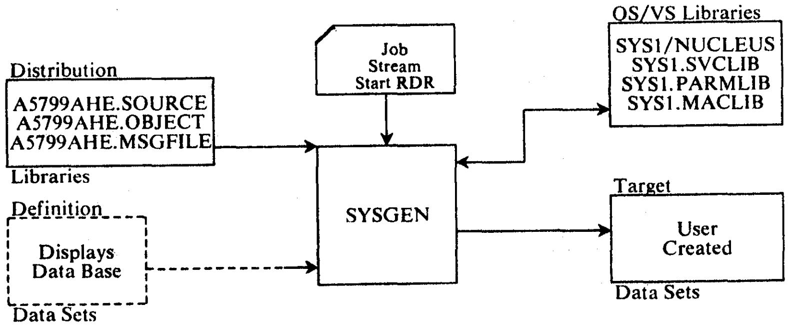


Figure 3-3. The Special Real Time Operating System SYSGEN - Stage II

One of the final steps of Stage II invokes the offline utility program. At this point the system-defined data base macros are processed, followed by the customer definition data base macros (DBDSET=). Then the system messages and the system displays (if Display Management is being generated) are processed. Following this, the customer-defined displays (DISDSET=) are processed by the offline utility.

Warning: The data base data sets are either partitioned or direct organization, and are built by the offline utility DPPXUTIL. The records and members of these data sets contain references to and have dependencies on other records and members. These references and dependencies are constructed by DPPXUTIL, and the data sets must not be modiifed except by DPPXUTIL. Concatenation of data base groups for realtime execution is not allowed. The macros used for SYSGEN and the available options are described in a following section.

SYSGEN RESTART PROCEDURES

The system generation process may come to an unsatisfactory completion because of errors that occurred during Stage I or Stage II. This section contains the information necessary to restart system generation.

The most common errors during Stage I and the restart procedures for Stage I are discussed, as are the most common error causes during Stage II, the restart techniques, and the reallocation of data sets.

The most common causes of error during Stage I are keypunching errors in the input deck and contradictory or invalid specifications in the macro instructions. Keypunching errors are indicated by system generation error messages or assembler error indications. Invalid specifications are indicated with the system generation error messages printed in the SYSPRINT data set. If any errors are found during Stage I, the job stream is not produced.

Stage I consists of a single assembly of the system generation macro instructions. It can be restarted only from the beginning. To restart Stage I, the errors in the input deck or in the definition data sets must be corrected and the job resubmitted.

The most common error causes during Stage II are:

- Machine interruptions and non-continuous machine time
- Faulty space allocation of the system data sets during the preparation for system generation
- Errors in the input deck that cannot be detected during Stage I
- Procedural errors such as improper volume mounting

Stage II can be restarted at the beginning of any job step. If any statements in the job stream are to be changed, the job stream must be on cards. If no statements are to be changed, the IEBEDIT utility program can be used to restart a job stream. A later section discusses the techniques used for restarting the job stream after any other necessary operations have been performed. The topics include restarting from cards, punching the job stream, and restarting from tape or from a direct access volume.

If the job stream is on cards, a job step can be restarted by placing a JOB card ahead of the job step's EXEC card and entering the cards in the card reader.

If the output from Stage I was not a card punch, the IEBPTPCH utility program can be used to punch the job stream. The following example shows the statements required to punch the job stream using IEBPTPCH. The fields shown underlined may require modification for different installations.

```
//PUNCH         JOB
//              EXEC     PGM=IEBPTPCH
//SYSUT1        DD       UNIT=182,LABEL=(,NL),VOLUME=SER=EXLABL,
//                       DISP=OLD,DCB=(RECFM=F,BLKSIZE=80)
//SYSUT2        DD       UNIT=2540-2
//SYSPRINT      DD       SYSOUT=A
//SYSIN         DD       *
    PUNCH                TYPORG=PS

/*
```

When using the IEBPTPCH utility program to punch the job stream, the following points should be considered.

- The value of the UNIT parameter of the SYSUT1 DD statement is the specific unit address of the magnetic tape drive or direct-access storage device on which the job stream resides. Unless the job stream tape or direct-access volume has been demounted, the value of this UNIT parameter is the same as the value of the UNIT parameter of the SYSGO or SYSLIN DD statement in the input deck for Stage I. If the job stream is on a direct access volume, the LABEL parameter must specify a standard label, and a DSNAME parameter must be specified.

- The value of the VOLUME parameter of the SYSUT1 DD statement is either an external serial number assigned to the job stream tape reel, or the volume serial number of the tape or direct access volume. The system will issue a MOUNT command for the specified volume on the magnetic tape or direct access storage device indicated by the UNIT parameter.

- Sequence numbers can be specified for the punched cards by putting the CDSEQ or CDINCR parameters in the PUNCH control cards of the IEBPTPCH input deck.

The IEBEDIT utility program can be used to restart Stage II from any job step, after the first, when the job stream is on tape or a direct-access volume. To restart form the first job step, a START RDR command can be issued for the tape drive or direct-access storage device that contains the job stream.

IEBEDIT creates a new job stream by editing and selectively copying the job stream provided as input. The IEBEDIT utility program can copy an entire set of jobs including JOB statements and associated job step statements, or selected job steps in a job, as shown below in the control statements required by IEBEDIT when the job stream is on tape.

```
//RESTART       JOB
//              EXEC    PGM=IEBEDIT
//SYSPRINT      DD      SYSOUT=A
//SYSUT1        DD      UNIT=xxx,LABEL=(,NL),                  X
//                      VOLUME=SER=serial,                     X
//                      DISP=(OLD,KEEP),                       X
//                      DSN=data set name,                     X
//                      DCB=(DCB information)
//SYSUT2        DD      UNIT=xxx,LABEL=(,NL),                  X
//                      VOLUME=SER=serial,                     X
//                      DISP=(,KEEP),                          X
//                      DSN=data set name,                     X
//                      DCB=(DCB information)
//SYSIN         DD      *
    EDIT        START=SYSGENnn,STEPNAME=SGxx(,NOPRINT)
or  EDIT        START=SYSGENnn,TYPE=INCLUDE,
                STEPNAME=(SGxx(,SGxx)...)(,NOPRINT)
or  EDIT        START=SYSGENnn,TYPE=EXCLUDE,
                STEPNAME=(SGxx(,SGxx)...)(,NOPRINT)
/*
```

When using the IEBEDIT utility program to restart Stage II, the following should be considered.

- The value of the UNIT parameter of the SYSUT1 DD statement is the unit address of the magnetic tape drive or direct-access storage device on which the job stream tape or direct-acess volume is mounted. Unless the job stream has been demounted, the value of the UNIT parameter is the same as the value of the UNIT parameter of the SYSGO or SYSLIN DD statement in the Stage I input deck. If the job stream is on a direct-access volume, the LABEL parameter must specify a standard label.

- The value of the VOLUME parameter of the SYSUT1 DD statement is either any serial number assigned to the job stream tape reel, or the volume serial number of the tape or direct-access volume. The system will issue a MOUNT command for the specified volume on the magnetic tape drive or direct-access storage device indicated with the UNIT parameter.

- The value of the UNIT parameter of the SYSUT2 DD statement is the unit address of a magnetic tape drive or direct-access storage device. If the job stream is on a direct-access volume, the LABEL parameter must specify a standard label.

- One or more EDIT statements can be specified when executing IEBEDIT. If the TYPE parameter is omitted, STEPNAME specifies the first job step in the job specified by the START parameter to be placed in the new job stream.

- If TYPE=INCLUDE or TYPE=EXCLUDE is specified, STEPNAME specifies the job steps to be included or excluded, respectively, from the new job stream. Individual job steps and sequences of job steps can be specified for inclusion or exclusion. For example:

  ```
  START=SYSGEN4,TYPE=INCLUDE,STEPNAME=(SG3,SG6-SG9)
  ```

  indicates that job steps 3, 6, 7, 8, and 9 of job 4 are to be included in the restart of system generation.

- NOPRINT must be included if a listing of the new job stream is not desired. After the new job stream is created, a START RDR command must be issued for the magnetic tape drive or direct-access storage device designated by the SYSUT2 DD statement.

An IEBEDIT input deck for restarting Stage II is shown below. In this example, space allocation for SYS1.SVCLIB was not sufficient, causing the subsequent job steps to fail.

```
//RESTART       JOB
//              EXEC    PGM=IEBEDIT
//SYSPRINT      DD      SYSOUT=A
//SYSUT1        DD      UNIT=2400,LABEL=(,NL,),DSN=STAGE,     X
//                      VOL=SER=JOBSTM,DCB=(RECFM=F,          X
//                      BLKSIZE=80,DEN=2),DISP=(OLD,KEEP)
//SYSUT2        DD      UNIT=2400,DISP=(,KEEP),               X
//                      VOL=SER=001234,DSN=OUTTAPE,           X
//                      LABEL=(,NL),                          X
//                      DCB=(RECFM=F,BLKSIZE=80,DEN=2)
//SYSIN         DD      *
    EDIT                START=S37001,TYPE=EXCLUDE,            X
                        STEPNAME=(SG1-SG24)
/*
```

The following section gives guidelines for restarting Stage II. Restarting may require the scratching and reallocation of space for the system data sets. When this is necessary, the following guidelines should be referenced for the procedure to be followed. After the necessary corrections have been made, the actual restarting of Stage II can be accomplished by one of the methods described.

If the problem encountered is other than space allocation, e.g., component failures or machine malfunctions, the instructions printed out in the error messages or error codes should be followed.

The method for reallocating space for a system data set depends on whether the data set contains data that must be saved. If the data set does not contain data that needs to be saved (for example, the data set will be re-copied completely when system generation is restarted), the IEHPROGM utility program can be used to scratch and reallocate space for the system data set. If the system data set contains data that must be saved, the data will have to be copied into a temporary data set, space for the original data set will have to be reallocated, and the contents of the data set will be copied from the temporary data set into the reallocation data set.

The input deck for scratching and reallocating space for system data sets must contain the following statements in the order shown:

1. A JOB statement with any parameters required by the particular installation

2. An EXEC statement with the PGM=IEHPROGM parameter

3. A SYSPRINT DD statement defining the system output unit

4. A DD statement defining the unit address and serial number of the generating system's system resident volume:

   ```
   //SYSRES      DD  UNIT=unit,VOLUME=SER=serial,DISP=OLD
   ```

5. A DD statement defining any other permanent volume on which the system data sets to be reallocated reside:

```
//OTHERVOL  DD  UNIT=unit,VOLUME=SER=serial,        X
                DISP=OLD
```

6. A DD statement for each type of removable volume on which the system data sets to be reallocated reside:

```
//DDNAME      DD  UNIT=(unit,,DEFER),               X
                      VOLUME=PRIVATE,DISP=OLD
```

7. A DD * statement (SYSIN)

8. A SCRATCH statement for each new system data set to be reallocated. The SCRATCH statement must have the following format:

```
SCRATCH DSNAME=dsname,VOL=device=serial,PURGE
```

9. A /* statement

10. An EXEC statement with the PGM=IEHPROGM parameter

11. A DD statement defining the unit address and serial number of the generating system's system residence volume (example shown above)

12. A DD statement for each permanent volume on which the system data sets to be reallocated reside (example shown above)

13. A DD statement for each type of removable volume on which the system data sets to be reallocated reside (example shown above)

14. A SYSPRINT DD statement defining the system output unit

15. A DD statement for each of the new system data sets to be reallocated. This DD statement must be the same as the one used in the input deck for the original allocation.

```
//ddname DD DSNAME=dsname,                          X
//              VOLUME=(,RETAIN,SER=serial),        X
//              UNIT=unit,LABEL=EXPDT=99350,        X
//              SPACE=(allocation),DISP=(,KEEP),    X
//              DCB=(parameters)
```

16. A DD * statement (SYSIN)

17. A /* statement

If the system data set to be reallocated contains data, one of two procedures can be followed. If there is enough space on the volume for a new space allocation, the following procedure may be used.

1. Rename the system data set.

2. Allocate space for the system data set (with its correct name) on the same volume using the IEHPROGM utility program.

3. Copy the data in the renamed data set onto the newly allocated system data set using the IEBCOPY utility program.

4. Scratch the renamed data set using the IEHPROGM utility program.

The following statement illustrates space reallocation for a data set on the same volume. The system data set to be reallocated is SYS1.PARMLIB. It was allocated space during the preparation for system generation with the following IEHPROGM DD statement:

```
//PARMLIB      DD      DSNAME=SYS1.PARMLIB,                       X
//                     VOLUME=(,RETAIN,SER=SYSTEM),               X
//                     UNIT=2314,DISP=(,KEEP),                    X
//                     SPACE=(TRK,(7,,3),,CONTIG),                X
//                     LABEL=EXPDT=99350,                         X
//                     DEB=(RECFM=F,BLKSIZE=80)
```

The new system residence volume is 2314 volume whose serial number is SYSTEM. The renamed SYS1.PARMLIB will be called SYS1.TEMPPARM.

```
//MOVE         JOB
//STEP1        EXEC    PGM=IEHPROGM            -RENAME-
//SYSPRINT     DD      SYSOUT=A
//NEWRES       DD      UNIT=2314,VOLUME=SER=SYSTEM,DISP=OLD
//SYSIN        DD      *
               RENAME  DSNAME=SYS1.PARMLIB,                       X
                       VOL=2314=SYSTEM                            X
                       NEWNAME=SYS1.TEMPPARM
/*
//STEP2        EXEC    PGM=IEFBR14             -REALLOCATE-
//PARMLIB      DD      DSNAME=SYS1.PARMLIB,                       X
//                     VOLUME=(,RETAIN,SER=SYSTEM),               X
//                     UNIT=2314,DISP=(,KEEP),                    X
//                     SPACE=TRK,(8,,3),,CONTIG),                 X
//                     LABEL=EXPDT=99350,                         X
//                     DCB=(RECFM=F,BLKSIZE=80)
/*
//STEP3        EXEC    PGM=IEBCOPY             -COPY-
//SYSPRINT     DD      SYSOUT=A
//SYSUT1       DD      DSNAME=SYS1.TEMPPARM,DISP=OLD              X
//                     UNIT=2314,VOL=SER=SYSTEM
//SYSUT2       DD      DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN        DD      *
    COPY               INDD=SYSUT1,OUTDD=SYSUT2
/*
//STEP4        EXEC    PGM=IEHPROGM            -SCRATCH-
//SYSPRINT     DD      SYSOUT=A
//NEWRES       DD      UNIT=2314,VOLUME=SER=SYSTEM,DISP=OLD
//SYSIN        DD      *
               SCRATCH DSNAME=SYS1.TEMPPARM,                      X
                       VOL=2314=SYSTEM,PURGE

/*
//
```

THE SPECIAL REAL-TIME OPERATING SYSTEM SYSGEN MACROS

The Special Real Time Operating System SYSGEN macros fall into two categories: configuration and software. The following pages define the SYSGEN macros and list the calling sequence for each.

CONFIGURATION CUSTOMER DIEFINTION DATA SET MACROS

CONFIGH

This macro defines configuration hierarchy. CONFIGH must be the first macro in each member of a configuration data set. For a Special Real Time Operating System only, it is not needed, and neither is the configuration data set. For a system with Display Management, it becomes the header macro for configuration information for the CPU it references.

symbol	CONFIGH	CPU=S370xx,LEVEL=integer

CPU

Must be of the form S370xx, where xx is a value between 01 and 99. For a system with the Special Real Time Operating System or the Special Real Time Operating System and Display Management, xx can be any value between 01 and 99.

LEVEL

Is a number between 01 and 99 which specifies at what level in the hierarchy this CPU occurs. A 1 indicates the top (highest) level. The value of this parameter increases by 1 each time a lower-level CPU is encountered.

The name of the configuration data set member containing the above macro must be the same as the CPU= parameter.

For a Special Real Time Operating System only, no other macros follow the CONFIGH macro if it is used. For a system with Display Management, DEFDEV macros follow the CONFIGH to define each display unit. Refer to the Display Management Description and Operations Manual for a description of this macro.

SOFTWARE CUSTOMER DEFINITION DATA SET MACROS

This section defines the various macros that can be placed in the members of a software options data set for the Special Real Time Operating System portion of a system generation. The name chosen for a member of the software option data set should be the same name used for the corresponding member of the configuration data set.

The macros defined below may appear in any order, except that the GENEMS macro must be last. All statements following the last continuation card of the GENEMS macro are ignored; as such, an assembler END statement is not required.

All of the macros are optional except the VS and GENEMS macro, which are required.

VS

Defines information relating to the customer's VS system.

[symbol]	VS	$\mathrm{MCS{=}(integer_1\,[,integer_2,\ldots,integer_n])}$ $\mathrm{,DESC{=}integer}$ $\mathrm{,SVCNO{=}(value_1,value_2,value_3)}$ $\left[\mathrm{,APNDG{=}(value_1,value_2)}\right]$ $\left[\mathrm{,NUCNUM{=}}\left\{\begin{matrix}\underline{1}\\ \mathrm{character}\end{matrix}\right\}\right]$ $\left[\mathrm{,CLOCKCP{=}}\left\{\begin{matrix}\underline{\mathrm{NO}}\\ \mathrm{YES}\end{matrix}\right\}\right]$ $\left[\mathrm{,RAM{=}xx}\right]$ $\left[\mathrm{,PTIME{=}}\left\{\begin{matrix}\underline{10}\\ \mathrm{number}\end{matrix}\right\}\right]$ $\left[\mathrm{,TIMEEXT{=}number}\right]\left[\mathrm{,TIMERAT{=}}\left\{\begin{matrix}\underline{60}\\ \mathrm{seconds}\end{matrix}\right\}\right]$ $\left[\mathrm{,GETWAS{=}(size,number[,size,number,\ldots])}\right]$ $\left[\mathrm{,TWOPART{=}}\left\{\begin{matrix}\underline{\mathrm{NO}}\\ \mathrm{YES}\end{matrix}\right\}\right]\left[\mathrm{,DIRSVC{=}}\left\{\begin{matrix}\underline{\mathrm{YES}}\\ \mathrm{NO}\end{matrix}\right\}\right]$

MCS

Is a list of integers, each with a value of 1 to 16, indicating which console routing codes are to be used by WTOs and WTORs issued within the Special Real Time Operating System.

DESC

Is a number from 1 to 9 indicating which descriptor codes are to be used by WTOs or WTORs issued within the Special Real Time Operating System.

SVCNO

Is three decimal integers, in the range of 200 to 255, indicating which user SVC members the customer has provided for the Special Real Time Operating System to use. The numbers are stated in Type I, Type II, Type IV order.

APNDG

Is required only if System/370 Energy Management System is being generated. It specifies the last two characters of the name to be used by System/370 Energy Management System for its I/O appendages and must meet the rules for user I/O appendages described in the publication OS/VS1 Data Management for Systems Programmer, GC28-0631.

CLOCKCP

If YES is specified, the optional PTIME use of the System/370 clock comparator feature is selected.

The CPU upon which the generated Special Real Time Operating System will be executed must have the clock comparator feature. The OS/VS1 system must be generated to not use the clock comparator feature.

NUCNUM

Is an alphameric character that specifies the eighth character of the OS/VS1 nucleus name to be created by generating the Special Real Time Operating System. IEANUC01 is always used as input to the Special Real Time Operating System generation. This parameter allows the output (modified) nucleus to be given a different member name. If NUCNUM is not specified, the resultant nucleus will have the name IEANUC01.

RAM

Specifies the seventh and eighth characters of the member to be created in SYS1.PARMLIB, which will contain a list of resident reentrant routines after the Special Real Time Operating System generation is completed. If this parameter is omitted, no list is created. If the data set specified in the LMDSET parameter of the GENEMS macro is concatenated with SYS1.LINKLIB via the LNKLST00 member of SYS1.PARMLIB, this RAM list member can be used to place the reentrant module of the Special Real Time Operating System (and Display Management and System/370 Energy Management System, if selected) in the link pack area.

PTIME

Specifies the time interval minimum value and basic cycle interval of PTIME. The default value is ten 10-millisecond units (100 ms). If a different Interval is desired, it must be specified as a number of 10-millisecond units.

GETWAS

Specifies the default sizes and number of blocks of each size to be reserved by GETWA at the Special Real Time Operating System initialization. The sizes must be specified in ascending sequence. It may be overridden at the Special Real Time Operating System initialization time.

The maximum number of sizes is 32. The maximum size allowed is 30720 bytes. The maximum number of blocks of a given size is 4095. Sizes greater than 2K must be defined as multiples of 2K.

Note: A GETWA space of sufficient size to satisfy the requirements of all Special Real Time Operating System programs must be provided. Failure to define sufficient GETWA space during system generation on the GETWAS parameter of the VS macro or on the GETWA statement in the SYSINIT input stream will result in the termination of the realtime job with a user 46 ABEND code. The Special Real Time Operating System routines require that blocks of at least 1024 bytes be defined.

TIMEEXT

Specifies on which external signal line (2-7) a periodic time pulse is available. This pulse is used to correct for long-term drift in the System/370 TOD clock. Its omission indicates that no time sync pulses are available.

TIMERAT

Specifies the period (in seconds) at which the periodic time pulse will occur. The default value is 60.

TWOPART

If YES is specified, a two-partition operation will be made available in the Special Real Time Operating System. If no (default) is specified, a two-partition operation will not be available. A two-partition operation should not be selected unless it is needed as it increases the size of the pageable nucleus.

DIRSVC

This parameter indicates how the Special Real Time Operating System macros which issue SVCs are to be expanded when the DCVTR and DCVTLOC parameters are not supplied. It applies only to the usage of the Special Real Time Operating System macros by user programs. The Special Real Time Operating System programs are required to use the DCVTR/ DCVTLOC parameter or to be assembled at SYSGEN time. If yes (default) is specified, the macro expansion will issue the correct SVC number inline. This ties the assembly of the user programs to the SVC numbers used at that installation. If no is specified, the macro will expand 6 load instructions to obtain the XCVT and then execute the SVC from the XCVT. Thus, the user program is not tied to the SVC numbers.

FAILRST

This macro causes the Failover/Restart facility to be included in the system. Also, it optionally includes the continuous monitor or PROBE and the Computer Status Panel.

[symbol]	FAILRST	$\left[\text{CONTMON=}\begin{Bmatrix}\underline{\text{NO}}\\\text{YES}\end{Bmatrix}\right]\left[\text{,CONTINT=}\begin{Bmatrix}\underline{1}\\\text{number}\end{Bmatrix}\right]$ $\left[\text{,CONTADL=(name}_1\left[\text{,name}_2\text{,...,name}_n\right]\text{)}\right]$ $\left[\text{,PROBE=}\begin{Bmatrix}\underline{\text{NO}}\\\text{YES}\end{Bmatrix}\right]\left[\text{,PROBIT=}\begin{Bmatrix}\underline{\text{LOW4}}\\\text{HIGH4}\end{Bmatrix}\right]$ $\left[\text{,EQUIPSW=}\begin{Bmatrix}\underline{\text{NO}}\\\text{number}\end{Bmatrix}\right]\left[\text{,EQUIPDY=number}\right]$ $\left[\text{,RESET=}\begin{Bmatrix}\underline{\text{NO}}\\\text{number}\end{Bmatrix}\right]$ $\left[\text{,STATUSP=}\begin{Bmatrix}\underline{\text{NO}}\\\text{YES}\end{Bmatrix}\right]\left[\text{,LTS=(n}_1\text{,n}_2\text{ [,n}_3\text{,n}_4\text{])}\right]$ $\left[\text{,FAILEXT=(number[,static line])}\left[\text{,CMCKPRB=}\begin{Bmatrix}\underline{\text{NO}}\\\text{YES}\end{Bmatrix}\right]\right]$

CONTMON
Causes the continuous monitor facility to be included in the system if YES is specified.

CONTINT
Specifies the period (in seconds) at which the continuous monitor is to check the operation of the online CPU and report to the backup CPU (if PROBE is selected).

CONTADL
Specifies the names of additional 2-byte virtual storage resident data base items which the continuous monitor is to periodically check. This is in addition to locations it implicitly checks within the Special Real Time Operating System.

PROBE
Causes the PROBE function to be generated if YES is specified. CONTMON=YES is required. The period at which the PROBE function expects to be transmitted to by the continuous monitor is specified in the CONTINT parameter.

PROBIT
Specifies whether the low-order (4-7) or high-order (0-3) bits of the direct control static data lines are to be used for the continuous monitor to send signals to the PROBE.

EQUIPSW
Specifies to which direct control signal-out line (0-7) the remote 2914 switch is attached. This option This option requires PROBE=YES.

EQUIPDY
Indicates how long (in milliseconds) the PROBE function is to delay after switching the 2914 before either IPLing the Failover/Restart data set or returning to allow the realtime job to continue. The delay is to allow the 2914 to complete the switch.

RESET

If specified, indicates on which direct control signal-out line (0-7) a signal can be sent to allow one CPU to system reset the other CPU. This option requires PROBE=YES.

STATUSP

If specified, indicates that the continuous monitor and/or PROBE is to support the Computer Status Panel. CONTMON=YES is required.

LTS

Is required if STATUSP=YES. Two values are required if PROBE=NO, and four values if PROBE=YES. The first (or only) two values indicate which direct control signal-out line (0-7) is to be used to illuminate the Online light and the Ready light. The second two values indicate which bits on the direct control static data lines (0-7) are used to illuminate the Failover Recommend and Computer Selected for Failover lights.

FAILEXT

If specified, the first parameter indicates which external signal line (2-7) will be used to indicate the Failover Confirmed Interrupt of the Computer Status Panel. Requires PROBE=YES and STATUSP=YES. The second parameter (optional) indicates which static signal line is used to verify that the Failover Confirmed External Interrupt is to be honored. This line must be 1 or the interrupt is ignored.

CMCKPRB

This parameter indicates if the PROBE function (backup CPU) is to be checked by the continuous monitor (online CPU). The PROBE (if selected) always checks the continuous monitor. If the continuous monitor detects that the PROBE is no longer running, it issues a message and continues operation.

Warning: If this option is chosen, the PROBE also writes and therefore it is possible to start a PROBE function in each CPU and have neither PROBE recommend failover as each PROBE is receiving data on the static data lines from the other PROBE. This is not possible without this option as the PROBE attempts only to "read" from the continuous monitor but never write to it.

DUPDISK

Includes duplicate disk data set support in the Special Real Time Operating System.

[symbol]	DUPDISK

DBASE

Specifies customer arrays to be generated.

[symbol]	DBASE	USERARR=(name$_1$,name$_2$,...,name$_n$)

USERARR
Specifies the member names of customer-supplied data base arrays, in source format, which are to be processed through the offline utility during Stage II SYSGEN. The data set containing the array definitions is defined in the DBDSET parameter of the GENEMS macro.

LOG

Includes data base logging in the Special Real Time Operating System.

[symbol]	LOG	LOGFREQ=(value ,value ,value)

LOGFREQ

Specifies the logging period in seconds corresponding to LOGFREQ values of 1, 2, and 3 in the ARRAY macro. All three values are required. The values must be in ascending sequence.

PLISUB

Indicates that PL/I structures and library routines are to be included for the Special Real Time Operating System services. If Display Management and/or System/370 Energy Management System are being generated also, structures and library routines are included for their services as well. These routines can be used with PL/I F, the PL/I Optimizing Compiler and the PL/I Checkout Compiler.

[symbol]	PLISUB

FORSUB

Indicates that FORTRAN library routines are to be included for the Special Real Time Operating System services. If Display Management and/or System/370 Energy Management System is being generated also, library routines are included for their services as well. These routines can be used with the FORTRAN G and H compilers.

[symbol]	FORSUB

MSGRC

Defines devices for routing codes for system messages.

[symbol]	MSGRC	RC=code [,ALTRC= {1 | code}] ,DEV= { SYSCONS (OSDEVICE,DDNAME=name) (DISPLAY,ACCESSA=name [,FUNCA=name]) (PATCH,EP=name) }

RC
Indicates which routing code is being fully or partially defined. Valid codes are numeric in the range of 1 to 255. Codes 1 through 9 are reserved for the Special Real Time Operating System. The customer should define the destination for codes 1 through 9 for the Special Real Time Operating System messages as well as his own from 10 to 255. A routing code can be defined to go to multiple devices by including multiple MSGRC macros with the same RC specification. The MSGRC macros must be in ascending routing code order. Code 1 will always go to the system console (in addition to any other defined devices).

ALTRC
Indicates an alternate routing code to use if the device defined is not available.

DEV
Indicates which device to output the message.

SYSCONS
Indicates that a WTO will be issued.

OSDEVICE
Indicates that a QSAM PUT will be done to the DDname specified.

DISPLAY
Is valid only in systems with Display Management. The message will be written to the system message zone using the indicated access area and function area codes, if supplied.

PATCH
Indicates that the message is to be passed to a Special Real Time Operating System independent task at the entry point indicated. The task name is the same as the entry point name.

ACCESSA
Indicates the Display Management access area associated with a DISPLAY routing code.

FUNCA
Indicates the Display Management function area associated with a DISPLAY routing code.

IMP

Indicates input message processing commands in addition to those defined as part of the Special Real Time Operating System. There is one IMP macro per code.

[symbol]	IMP	CODE=name,TASK=name,LM=name, $\left[,\text{ID=}\left\{\begin{matrix}\underline{0}\\ \text{number}\end{matrix}\right\}\right]\left[,\text{PARAM=}\left(\text{val}_1\left[,\text{val}_2,\ldots,\text{val}_n\right]\right)\right]$

CODE
Is the command which the Input Message Processor is to recognize; it contains a maximum of eight characters. By specifying a command implicitly defined by the Special Real Time Operating System the customer can re-define these codes.

TASK
Is the name of the task which is to be PATCHed as a result of the command being entered.

LM
Is the load module name which is to be PATCHed.

ID
Is the ID field to be passed to the PATCHed task.

PARAM
Indicates the conversion codes of positional parameters that will be passed to the task. Each value is of the form Tl, where T can be C(character), X(hexadecimal), or F(fixed point decimal); l represents the length of the area into which the data is converted; l can be any values from 1 to 255.

DATA SET

Indicates location of noncataloged OS/VS1 data set. If the OS/VS1 data set is cataloged, this macro need not be specified.

[symbol]	DATASET	name,VOL=(serial,type)

name
Is a positional parameter that indicates for which OS/VS1 data set location information is being given. Valid values are:

NUCLEUS
SVCLIB
MACLIB
PARMLIB
TELCMLIB

VOL
Indicates the volume serial number and device type upon which the data set in question resides, e.g., VOL=(TST346, SYSDA).

GENEMS

Generates the Special Real Time Operating System.

```
[symbol]   GENEMS    CPU= { 370    }
                          { S370xx }
                    [,ASMPRT= { ON  } ][,ASMBLR= { F } ]
                    [         { OFF } ][         { H } ]
                    [,LKPRT= ([{ MAP  }][,LIST]))]
                    [        ([{ XREF }][     ]))]
                     ,JOBCTL=([{    A     }][{    A      }][,job acct][,step acct])
                             ([{ jobclass }][{, outclass }][         ][          ])
                     ,OBJDSET=name  [,OBJVOL=(serial,type)]
                     ,LMDSET=name   [,LMVOL=(serial,type) ]
                     ,MACDSET=name  [,MACVOL=(serial,type)]
                    [,DBD         ] [,DBVOL=(serial,type) ]
                    [,DISDSET=name] [,DISVOL=(serial,type)]
                     ,ARRDSET=name  [,ARRVOL=(serial,type)]
                     ,DB1DSET=name  [,DB1VOL=(serial,type)]
                     ,DB2DSET=name  [,DB2VOL (serial,type)]
                    [,DB3DSET=name] [,DB3VOL=(serial,type)]
                     ,DB4DSET=name  [,DB4VOL=(serial,type)]
                    [,DB5DSET=name] [,DB5VOL=(serial,type)]
                    [,PLIDSET=name] [,PLIVOL=(serial,type)]
                    [,PLSDSET=name] [,PLSVOL=(serial,type)]
                    [,FORDSET=name] [,FORVOL=(serial,type)]
                    [,OS2DSET=name] [, OS2VOL=(serial,type)]
```

CPU
May be specified as either S370 or S370xx, where xx is equal to the ID assigned to this CPU in the CONFIGH macro CPU keyword.

ASMPRT
Is indicated if assembly listings are to be produced during Stage II. The default is OFF.

ASMBLR
Indicates which assembler is to be used during Stage II. The default is the OS/VS1 Assembler (F). The Assembler H Program Product, 5734-AS1, may be specified. If the H assembler is specified, it is assumed that it has been installed using the default DD names.

LKPRT
Indicates which linkage editor listing options are desired.

JOBCTL
Indicates values for job and SYSOUT classes and accounting information for the Stage II job stream.

jobclass
Specifies the value to be used in the CLASS parameter of the generated job card.

outclass
Specifies the output class to be used in the SYSOUT parameter on DD cards and the MSGCLASS parameter on the JOB card.

job acct
Is the information to be reproduced in the accounting field of the JOB card.

step acct
Is step accounting information to be reproduced in the ACCT parameter of each EXEC card.

The following table summarizes the use of each XXXDSET parameter. The value specified is in each case the name of a data set allocated and named by the installing installation. The corresponding XXXVOL parameter is used to indicate the location of the data set, e.g., XXXVOL= (TST346,SYSDA), if it is not cataloged. All of the data sets must be disk resident, and all are partitioned except the DB2DSET which is direct organization and the OS2DSET which is sequential or the member of a partitioned data set.

Parameter	Required	Contents	DCB Info

OBJDSET	Yes	Output of Language Translator During Stage II	LRECL=80,RECFM=FB,
			BLKSIZE=XXX[1]

LMDSET	Yes	Load Modules for real Time Execution	RECFM=U,BLKSIZE=XXX[2,6]

MACDSET	Yes	Macros Generated During Stage II and For Customer Use	LRECL=80,RECFM=FB,
			BLKSIZE=XXX[3]

DBDSET	If USERARR PARM in DBASE Macro is Used	Customer-Defined Data Base Arrays	LRECL=80,RECFM=FB,
			BLKSIZE=XXX[3]

DISDSET	If DISM in System and User Displays Defined	Customer-Defined Display Definition	LRECL=80,RECFM=FB,
			BLKSIZE=XXX[3]

ARRDSET	Yes	Arrays Generated During SYSGEN	LRECL=80,RECFM=FB,
			BLKSIZE=XXX[3]

DB1DSET	Yes	Data Base Arrays	RECFM=U,BLKSIZE=XXX[2]
DB2DSET	Yes	Data Base Arrays	RECFM=U,BLKSIZE=XXX[2],DSORG=DA
DB3DSET	If DISM in System	Displays	RECFM=U,BLKSIZE=XXX[2]
DB4DSET	Yes	Messages	RECFM=U,BLKSIZE=292
DB5DSET	If DISM in System	Displays	RECFM=U,BLKSIZE=XXX[2]
PLIDSET	If PLISUB Macro Specified	PL/I Library Routines	RECFM=U,BLKSIZE=XXX[4,6]

PLSDSET	If PLISUB Macro Specified	PL/I Structures .	BLKSIZE=XXX[3,5]
			LRECL=80,RECFM=FB,
			BLKSIZE=XXX[3,5]

FORDSET	If FORSUB Macro Specified	FORTRAN Library Routines	RECFM=U,BLKSIZE=XXX[4,6]
OS2DSET	If installing in Rel 3.0 or later	OS/VS Stage II SYSGEN job stream	RECFM=FB,LRECL=80,

[1] Maximum of 3200 due to OS/VS Linkage Editor restriction.

[2] Value of at least half track length recommended.

[3] Should have same BLKSIZE as installations SYS1.MACLIB.

[4] May be same data set as LMDSET.

[5] If PL/IF is to be used, BLKSIZE cannot exceed 400.

[6] A value equal to SYS1.LINKLIB recommended. Minimum of 7294.

Figure 3-4. XXXDSET Parameter Values

SYSTEM INITIALIZATION

The Special Real Time Operating System executes as a job step under control of OS/VS1. The job is started initially through standard OS/VS1 Job Control (JCL) statements with the EXEC card specifying PGM=DPPINIT. The JCL defines to the Special Real Time Operating System the data sets which have been created by the offline utility and the Special Real Time Operating System SYSGEN procedures. The JCL also defines the devices such as display and data acquisition, which are to be used by the online routines. Control statements for the initialization of subsystems are defined to the Special Real Time Operating System through the //SYSINIT DD card. Also included in the SYSINIT input stream are certain Special Real Time Operating System parameters that can override SYSGENed values.

The Special Real Time Operating System initialization consists of three processing phases: card read, basic initialization, and subsystem initialization, as shown in Figure 3-5.

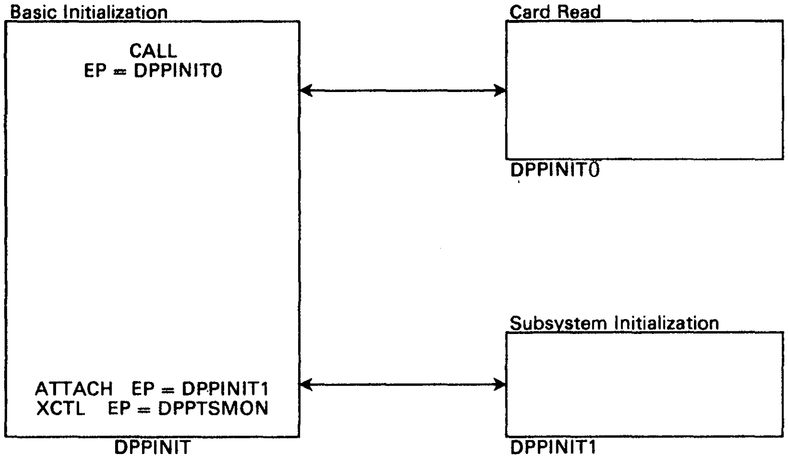


Figure 3-5. The Special Real Time Operating System Initialization

Program DPPINIT gains control from OS and immediately CALLs program DPPINITO, the control statement read routine. DPPINITO reads control statements from the input stream specified by the DD card named SYSINIT, and builds a chain of control blocks to represent the input stream, with one block built for each PATCH, WAIT, RESTART, and ABEND card found in the input stream. When End-of-File (EOF) is reached, control is returned to DPPINIT, with register 1 containing the origin of the control block chain. DPPINIT initializes the task management control blocks and when this is completed, attaches program DPPINIT1, then XCTLs to DPPTSMON. DPPINIT passes the origin of the control block chain built by DPPINITO to DPPINIT1, which processes and issues the PATCHes as specified by the user in the input stream. Figure 3-6 shows the control statements that are valid as input to initialization.

The control statement input stream defines the sequence of events that is to occur during subsystem initialization. The stream is a series of card image input statements coded similar to assembler language macros. The rules for continuation of control statements are the same as those for continuation of assembler language macro calls.

A control statement consists of a NAME field which is optional, an OPERATION field, which is required, and operands. The maximum number of operand characters is 255. There is no limit on the number of continuation statements, as the limiting factor is the number of characters of operands.

NAME	OPERATION	OPERANDS
label	PATCH	EP=name [,TASK=name] [,QL=n] [,ID=n] $\left[\text{,PRTY =}\left\{\begin{matrix}\text{JOBSTEP-n}\\\text{(taskname,n)}\end{matrix}\right\}\right]$ $\left[\text{,PARAM=}\left(\left\{\begin{matrix}\text{C'characters'}\\\text{X'hexadecimal digits'...,}\\\text{F'decimal number'}\end{matrix}\right\}\right)\right]$
label	WAIT	label
label		
label	QH	name ,QL=$\frac{255}{n}$,SEQ= $\frac{NO}{YES}$,HOLD= $\frac{NO}{YES}$,PATCH= $\frac{YES}{NO}$
label	QP	number, QH=(name$_2$,...,name$_n$) , PRTY= $\frac{JOBSTEP-5}{JOBSTEP-n}$,HOLD= $\frac{NO}{YES}$
label	STAEX	EXIT=name, LM=(name$_1$,name$_2$,...,name$_n$)
label	RESTART	$\left\{\begin{matrix}\text{,WRITE}\\\text{,}\underline{\text{NOWRITE}}\end{matrix}\right\}$ $\left\{\begin{matrix}\text{,PROBE}\\\text{,}\underline{\text{NOPROBE}}\end{matrix}\right\}$ $\left\{\begin{matrix}\text{,CMON}\\\text{,}\underline{\text{NOCMON}}\end{matrix}\right\}$ $\left\{\begin{matrix}\text{,CANCEL}\\\text{,}\underline{\text{NOCANCEL}}\end{matrix}\right\}$
label	TCB	number
label	GETWA	(number,value,...)
label	CBGET	number
label	ABEND	t,DUMP
label	MASTER	SLAVE=jobname
label	SLAVE	MASTER=jobname
*	comment	comment
label	DBREF	$\left\{\begin{matrix}\underline{\text{YES}}\\\text{NO}\end{matrix}\right\}$

Figure 3-6. Control Statement Input Stream

PATCH
Causes the creation of a task as defined by the PATCH macro. The PATCH control statement operands are:

EP=name
Is the name, one to eight characters in length, of the program to be PATCHed. EP= must be specified. The card read routine checks that the name does not exceed eight characters, but does no other validity checking on the name. This applies to any other operand that requires name, taskname, or jobname.

TASK=name
Is the name, one to eight characters in length, to be given to the task created by this patch.

QL=n
Is the maximum number of work queue entries to be given to the task. The number (n) must be a decimal number from 0 to 255, with a default value of 1.

ID=n
Must be a decimal number from 0 to 255 which will be passed to the PATCHed program. The default is 0, and 255 has special meaning as specified in the PATCH macro documentation.

PRTY=
Is used to determine the priority of the PATCHed task. When JOBSTEP-n is coded, the PATCHed task's priority is calculated by subtracting the value "n" from the highest priority available to users, which is the job step task (DPPTSMON) priority minus three. Value n must be a decimal number from 0 to 255. When (taskname,n) is coded, the task is given a priority of the task specified in taskname minus the value n. In either case, value n is a decimal value from 0 to 255. There is no default, and value n must be supplied. If PRTY is not specified, the task will have the priority of the PATCHor (in this case the highest possible user priority).

PARAM=
Specifies the parameters to be passed to the PATCHed program. There are three types of data which can be coded:

C'characters'
Cause a character string to be passed. The single quote character (') is not allowed in the character string.

X'hexadecimal digits'
Cause hexadecimal data to be passed and must be valid hexadecimal digits 0-9 or A-F.

F'decimal number'
Causes a fullword value to be passed and must be a signed decimal number from 0 to 2_{31}. A conversion will be done by the initialization program.

WAIT
Causes initialization to wait for the completion of a specified task, the task being the one PATCHed by the control statement with the same name as the operand "label." The wait is only for the task to return (i.e., the processing of the work queue that is created as a result of the patch card to be completed) and has no relationship to any work that may be created by that task via PATCHes or other means.

QP
Causes the creation of a queue processor identified by the number parameter and a logical sequence for processing queue holders defined by the QH parameter. The number and QH parameters are required. The QP control statement operands are defined as follows:

number
Is a numberic value from 0 to 99 used to uniquely identify a queue processor. If more than one QP statement is found in the input stream with the same number, initialization will be terminated. This queue processor may be referenced in subsequent QS commands by this number. An internal TCBX (task) name will be created for each QP in the format ****QPnn, where nn is the number specified here. PATCHes specifying the queue processor name as the task name will be rejected and a condition code will be returned to the user.

QH=name
Defines the names of from 1 to 21 queue holders for which work is to be processed by this queue processor. Any one queue holder name may be specified on up to 21 QP statements. The specified queue holder names are treated on a priority basis where work from the queue holder appearing first in the list will be processed first. This queue processor will select work from the first queue holder specified

until all work in the queue holder has been selected before selecting work from the second queue holder specified. Each queue holder name specified on a QP statement must be defined by a QH statement.

PRTY=

Is an optional parameter used to determine the dispatching priority of this queue processor task. When JOBSTEP-n is coded the queue processor task's priority is calculated by subtracting the value, n, from the highest priority available to users, which is the jobstep task (DPPTSMON) priority minus three. The value, n, is a decimal number from 0 to 255. If PRTY is not specified, the queue processor task will be assigned a priority of JOBSTEP-5 (i.e., the job step task's priority minus 8).

HOLD=

Is an optional parameter that can be used (HOLD=YES) to inhibit this queue processor from selecting work from any queue holder until released by a subsequent QS command specifying r xx,QS,QPnn,REL where nn is the number specified on this QP statement (or the equivalent using the ALL or ALL QP operands). If HOLD=YES is not specified, theis queue processor will be immediately available for normal processing.

QH

Defines the queue holders and identifies each by the name parameter. The name parameter is required. The QH control statement operands are defined as follows:

name

Is a 1 to 8 character name used to uniquely identify a queue holder. If more than one QH statement is found in the input stream with the same name, initialization will be terminated. This queue holder may be referenced in subsequent QS commands by this name. Work requests for this queue holder must specify this name as the task name on the PATCH -macro call.

QL=

Is an optional parameter that is used to limit maximum number of work queue entries to be given to this queue holder. This number, n, must be a decimal number from 1 to 255. The default queue length is 255.

SEQ=

Is an optional parameter that can be used (SEQ=YES) to request that work queued to this queue holder be processed sequentially (i.e., whenever work has been selected from this queue holder by a queue processor, no other queue processor may select work from this queue holder until that work has been completed) until altered by a subsequent QS command specifying r xx,QS,name,NONSEQ where "name" is the name specified on this QH statement. If SEQ=YES is not specified, work from this queue holder may be processed simultaneously by all queue processors which are eligible to process work from it.

HOLD=

Is an optional parameter that can be used (HOLD=YES) to prohibit any queue processor from selecting work from this queue holder until released by a subsequent QS command specifying r xx,QS,name,REL where "name" is the name specified on this QH statement. If HOLD=YES is not specified, this queue holder will be immediately available for normal processing.

PATCH=

Is an optional parameter that can be used (PATCH=NO) to cause all PATCHes to this queue holder to be rejected and a condition code to be returned to the user until altered by a subsequent QS command specifying r xx,QS,name,PATCH where "name" is the name specified on this QH statement. If PATCH=NO is not specified, this queue holder will accept all valid PATCHes.

STAEX
Is used to specify an exit routine load module that will be given control when one of the load modules specified on this STAEX statement abends. Multiple STAEX statements may be included in the SYSINIT input stream to define additional exit routine and/or load module names. However, if a particular load module name is specified on more than one STAEX statement, the exit routine defined on the last STAEX statement in the input stream that references this load module name is the exit routine that will be given control in the event of an abend of that load module. Unless the exit routine requests that the STAE processing be bypassed, the STAE options as defined by the STAE IMP command (i.e., DUMP, NODUMP, etc.) will remain in effect.

EXIT=
Is the name of an exit routine load module to be given control through standard linkage conventions during STAE processing. The same exit routine may be specified on two or more STAEX statements. This routine will be given control while in STAE processing and standard limitations for STAE routine apply (i.e., a STAE macro cannot be issued, etc.). On entry to the exit routines, registers 0, 1, 13, 14 and 15 will contain the values as defined by OS/VS1 STAE interface routines. Register 2 will contain the address of the QP TCBX. The exit routine must specify, by a return code in register 15, one of the following:

Return Code	Action to be Taken
Zero	Continue STAE processing as defined by the STAE IMP command (i.e., DUMP, NODUMP, etc.).
Positive Value	Bypass STAE processing and return to the OS/VS1 ABEND processing routine with registers 0, 1, and 15 as returned by the exit routine. This will allow the user to schedule a retry routine.
Negative 4	Bypass STAE processing, zero register 15, and return to the OS/VS1 ABEND processing routine. Abnormal termination will continue.

If the load module specified is not available on the JOBLIB/STEPLIB data sets at initialization time, initialization will be terminated.

LM=
Is the name of one or more user load modules for which the specified exit routine is to be given control in the event of an abend of that load module.

RESTART
The operands are not positional and may be coded in any sequence; however, if the statement is to be in the input stream, at least one of the operands must be used.

WRITE or NOWRITE specifies whether or not the failover data set is to be written.

PROBE causes the PROBE function to be PATCHed after the RESTART processing whether or not a failover data set is written. NOPROBE causes no PATCH to the PROBE. If both PROBE and CMON are requested, the PROBE is PATCHed first.

CMON causes the continuous monitor function to be PATCHed.

NOCMON does not PATCH the continuous monitor.

CANCEL causes the job step task to ABEND with a user code 45 immediately after the RESTART processing. The CANCEL function will occur prior to the PATCH to PROBE or continuous monitor.

NOCANCEL does not ABEND the jobstep and normal processing continues.

TCB
Causes a change in the number of advance TCBs to be obtained by initialization. The number (0-99) overrides the value specified at SYSGEN time. If more than one TCB statement is found, the value used will be the value on the last statement.

GETWA
Overrides the SYSGENed values for GETWA sizes. The values must be in parentheses and be paired (i.e., number,value). The maximum number of pairs is 32, where number represents the number of blocks, and value represents the size of the GETWA blocks; i.e., (5,72) requests 5 blocks of 72 bytes each. The maximum size of number is 4095, and the maximum size of value is 30710 bytes. If more than one GETWA statement is found in the input stream, the values used for initialization are the values from the last GETWA statement encountered. If two parameters request the same size, the second request is unusable. Sizes greater than 2K must be 2K multiples. The Special Real Time Operating System uses GETWA space in blocks up to 1024 bytes. If the GETWA statement is used, it must include blocks of 1024 bytes or larger.

CBGET
Causes the amount of CBGET (the Special Real Time Operating System Control Block) storage to be varied. The initialization default value is the number of TCBs multiplied by TCBXLNTH, rounded to 2K plus 6K. The value, specified by number, overrides the initialization default value and is a decimal number from 1 to 99 representing the number of 2K blocks of storage to get for CBGET core. For example, 10 would get 20K of CBGET storage. If more than one CBGET statement is in the input stream, the value used is the value from the last statement encountered. A CBGET 0 statement will cause initialization to use a default value for CBGET storage. This would be the same as if no CBGET statements were in the input stream.

ABEND
Is a control statement used in a testing environment. When an ABEND card is processed, the job step will be ABENDed with a user 22 ABEND after a time specified by t, where t is the number of seconds from 1 to 999. The default value for t is 30 seconds. A dump can be taken by coding DUMP, and the default is no dump. Control statements that follow the ABEND statement in the input stream will never be processed, as the ABEND causes a STIMER WAIT followed by the user ABEND.

MASTER
Is a statement used to designate this Special Real Time Operating System initialization as a MASTER partition for two-partition operation. SLAVE=jobname specifies the jobname of the SLAVE partition.

SLAVE
Indicates this initialization is for a SLAVE partition in two-partition operation. The MASTER-jobname operand specifies the jobname of the MASTER partition. Only one MASTER or SLAVE card is allowed in an input stream, and the jobname on the operand must be unique in the system.

*
 Is a comment statement. No continuations are allowed on comment statements and there is no limit to the number of comment statements in an input stream. Comments do not affect the initialization sequence, but will appear in the listing of control statements.

DBREF
 Indicates to data base logging that the data base should be refreshed during a normal start operation. That is, the most recently logged copy is to be used. The operand NO must be coded to stop data base refresh. The absence of a DBREF statement is the same as a DBREF YES. A DBREF NO statement in the input stream takes precedence over any DBREF YES statements in the same input stream.

The card read routine reads until EOF is reached and then returns control to DPPINIT. All control statements are processed, and input statements and any diagnostic error messages are written to the data set specified by the SYSPRINT DD statement. If any control statements are in error, the run is aborted with a user 34 ABEND, and a WTO message is written to the console.

The following example shows the JCL required and a typical input stream for the Special Real Time Operating System system initialization.

```
//REAL          JOB      3DE501,'PROGRAMMER',CLASS=F
//              EXEC     PGM=DPPINIT
//STEPLIB       DD       DSN=ACS370.LM01,DISP=SHR
//DBINIT        DD       DSN=ACS370.DB1,DISP=SHR
//DBINIT2       DD       DSN=ACS370.DB2,DISP=SHR
//MSGDS         DD       DSN=ACS370.DB4,DISP=SHR
//DPPFAIL       DD       DSN=ACS370.FALRST,DISP=OLD
//SYSPRINT      DD       SYSOUT=A
//MSGOUT        DD       SYSOUT=A
//SYSUDUMP      DD       SYSOUT=A
//SYSINIT       DD       *
P1              PATCH    EP=PINIT00,TASK=STARTER,                       *
                         QL=5,ID=7,PRTY=JOBSTEP-15
P2              PATCH    EP=XINIT,TASK=SUBSYS1,                         *
                         QL=10,ID=10,PRTY=JOBSTEP-16
P3              PATCH    EP=DBUILD,TASK=DATBAS,ID=255
W1              WAIT P2
P4              PATCH    EP=XUSE,TASK=SUBSYS2,                          *
                         QL=5,ID=15,PRTY=(SUBSYS1,5),                   *
                         PARAM=(F'47',F'52',X'A0B')
P5              PATCH    EP=PUSE
                RESTART  WRITE
P6              PATCH    EP=XREINT,TASK=MCTL,QL=15,                     *
                         PARAM=(C'POSTWRS')
P7              PATCH    EP=DBRST,TASK-DBUSE,                           *
                         PRTY=JOBSTEP-2
```

In this example, the JOB card is standard OS, and accounting information must be as required for the individual installation. The EXEC card must specify PGM=DPPINIT. The STEPLIB DD card points to the library(ies) containing the Special Real Time Operating System and user programs. The library name will depend upon the name given the data sets at SYSGEN time. The data sets required for the data base are pointed to by the DD cards DBINIT and DBINIT2. The online message handler requires the MSGDS and MSGOUT DD cards. The SYSPRINT DD card is required by initialization to print the input control statements. A SYSUDUMP or SYSABEND DD card is optional, depending on whether a dump is required on ABEND conditions. The SYSINIT DD card is required, and it must point to the data set containing the control statements for the online run.

The input control statements in the preceding example show a typical initialization sequence. The RESTART statement implies a wait on PATCH statements labeled P1, P2, P3, P4, and P5. There is also an implied wait on P6 before initialization is completed. The The RESTART WRITE makes the DPPFAIL DD card necessary.

All programs which are to be PATCHed via a PATCH statement prior to the RESTART statement must go to termination before the restart data set can be written. Each program PATCHed prior to the RESTART statement is PATCHed with the ECB= operand. The ECB will be posted with the POST bit, plus the contents of register 15 at the time the PATCHed program returns control to the Special Real Time Operating System. If there is no RESTART WRITE in the input stream, there is an implied wait before initialization termination on all PATCHes issued with the PARAM= keyword coded on the PATCH statement. Any PATCH receiving a non-zero return code will cause the job step to abend with a user 031 ABEND code.

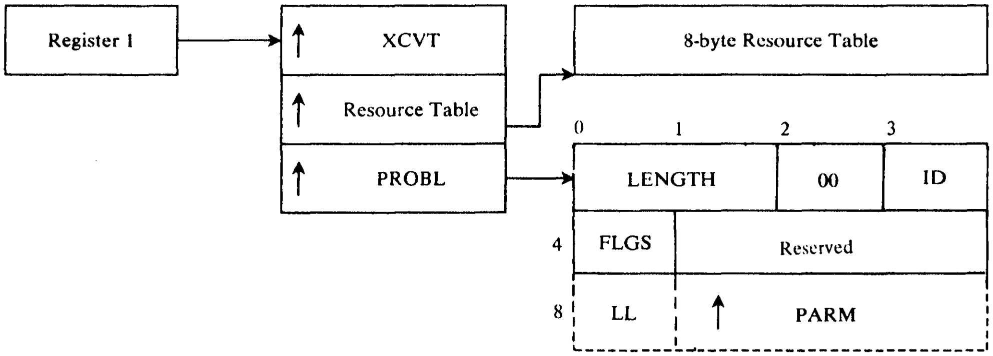
PATCHes in the input stream that follow a RESTART statement do not imply WAIT unless the PATCH statement contains the PARAM= keyword. There is an implied wait on each PATCH with the PARAM= keyword that follows the RESTART WRITE statement. Explicit waits may be forced on any PATCH statement through the use of the WAIT statement.

Upon regaining control from DPPINITO, DPPINIT initializes the task management control blocks. The XCVT and SCVT are initialized in subpool 253. The MASTER and SLAVE partitions are synchronized at this point if the run is for two-partition operation. DPPINIT then creates the TMCT in subpool 253 and initializes the GETWA control blocks and GETWA core, and the Special Real Time Operating System control block (CBGET) core. After creating the advance TCBs, DPPINIT links to other Special Real Time Operating System initialization routines in the following order:

- Duplicate Data Set Support (if SYSGENed)
- Data Base
- Realtime Message Handler
- Time Management
- Data Base Logging

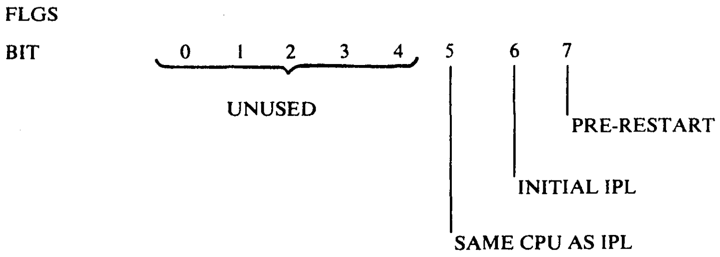
Upon completion of these routines, task management is initialized and ready to process PATCHes. At this time, DPPINIT attaches DDPINIT1 and then XCTLs to DPPTSMON.

Program DPPINIT1 processes the input stream and PATCHes the subsystem programs. A program that has been PATCHed by initialization receives control with pointers as shown below.



The PROBL contains the LENGTH, which is the length of the PROBL including parameter pointers. If PARAM= were coded on the PATCH input statement, there would be one word appended to the PROBL for each parameter being passed. The format of this word is that the high-order byte contains the length of parameter data, and the low-order three bytes contain the address of the data.

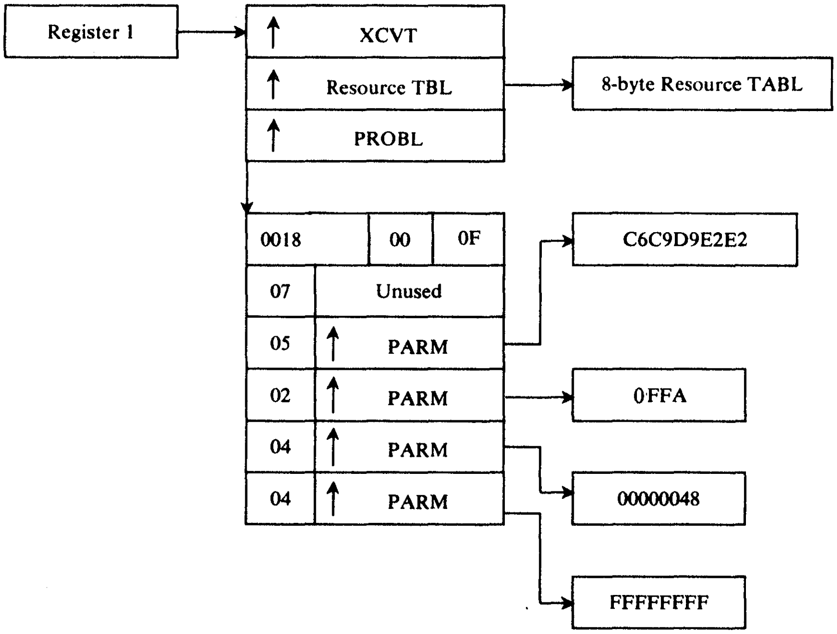
The ID is the value coded in the ID= field of the PATCH statement. The FLGS are as shown below:



Through interpretation of the PROBL FLGS, programs can determine if they were PATCHed prior to the writing of the failover data set (bit 7=1), if the system is being restarted (bit 6=0), or if the system has been failed-over to a backup CPU (bit 5=0).

The following PATCH statement would cause the program named REFNAME to receive control with parameters as shown below.

```
P1      PATCH    EP=REFNAME,ID=15,
                 PARAM=(C'FIRST',X'FFA',F'72',F'-1')
```



PATCHes issued prior to RESTART WRITE are issued with ECB=, and upon completion the post code is checked. MESSAGE DPP044I is issued for each PATCH prior to RESTART WRITE that is posted with a non-zero post code. If any ECBs have been posted non-zero, the job step will be ABENDed with a user 35 ABEND code just prior to the writing of the restart data set.

PATCHes issued for PATCH statements following the RESTART WRITE statement will be issued with the ECB= keyword also, only if they are coded with PARAM= or have WAIT statements naming the PATCH statement. As prior to RESTART WRITE, the ECBs are checked for non-zero post code, and message DPP044I will be issued for any task being posted non-zero. The job step, however, will not be ABENDed due to post codes for PATCHes following RESTART WRITE. If there is no RESTART WRITE statement in the input stream, the job step will be ABENDed (user 35) for any task posted non-zero.

When all the input stream control blocks have been processed, DPPINIT frees all storage obtained, and exits from the system. At this point Special Real Time Operating System and subsystem initialization is completed.

DDS Initialization

The initialization of Duplicate Data Set (DDS) support is accomplished by including the DUPDISK macro in the Special Real Time Operating System SYSGEN input. This will cause the initialization to link to DDS initialization in the prescribed sequence.

DDS initialization consists of the following functions:

1. Processing the DDS input control stream (defined by DDSCTLIN DD card) for DDS declarations.

2. Initially writing (and creating if not already done) the DDSTATUS data set record (calculating the maximum block size for use in later updates to this data set).

3. Allocating a DDS control header table (DDSCTLHD) and one DDS control area (DDSCTLA) for each DDS declared (initializing these tables with correct values).

4. Defining locks for each DDS declared (logically referred to as DDS-lock/share-ECB-chain locks).

5. Loading all DDS load modules (except the large and infrequently used modules) and saving their addresses in the DDS control table header.

6. Connecting the DDS control table header to the SCVT and each DDS control area to the DDS control table header.

Detailed Explanations

The DDS input control stream (consisting of 80-byte card images) is outlined in the following table:

Name	Opcode	Parameters
ddsname	DDSNAMES	(ddname$_1$,ddname$_2$[=OUT])
blank	REFRESH	blank
blank	READONLY	blank

DDSNAMES
This op code is used to declare a data set pair as being duplicates. There must be one DDSNAMES card for each data set pair the user wishes to be treated as duplicates. The number of declarations cannot be changed after initialization.

ddsname
Is the name which must be used by all DDS macros and commands which refer to this DDS. The The DDSDCB must use this name in its name field. If this operand is omitted, the value specified as ddnamel will be taken as the DDSNAME.

ddname1
This parameter is required and specifies the DDNAME of the primary data set.

ddname2
This parameter is required and specifies the DDNAME of the backup data set.

=OUT
Specifies that the backup should be initialized out-of-service.

REFRESH

This op code indicates that a DDSTATUS data set has already been created and that the declarations contained therein should be used for this run. This op code is only valid as the first card, and, if present, all subsequent cards will be ignored.

READONLY

This op code indicates that this is the backup computer and that all DDS outputs should be inhibited until failover/restart occurs. This op code implies REFRESH, so a previously created DDSTATUS data set is required. This op code is only valid as the first card, and all subsequent cards will be ignored.

The DDSTATUS data set will be sequential and will consist of one record (undefined record format) which will be the core image of the DDS control areas for all duplicate data sets. Thus, the needed information is contained for each DDS; primary DD name, backup DD name, and serviceability of the backup. This data set allows DDS to continue using the most current status for each DDS after a failover/restart.

DDS lock/share logic is required since more than one task may be using a DDS during the same time period. Some DDS functions require that no other tasks be using that same function at the same time, while other functions can proceed in parallel with each other. Thus four logical states can be defined for tasks with respect to DDS: (1) locking a DDS, (2) waiting-to-lock a DDS, (3) sharing a DDS, and (4) waiting-to-share a DDS.

The implementation of these states is accomplished by having a chain of DDS Lock ECBs and DDS share ECBs, both starting with the DDS control area for each DDS. The DDSLOCK ECB chain will consist of all tasks waiting-to-lock this DDS. A non-zero value in the high-order byte of the starting DDS lock ECB signifies that DDSLOCK is in effect for that task (as opposed to 'waiting-to-lock'). The DDS share ECB chain consists of all tasks waiting to share this DDS. The high-order byte of the starting DDS share ECB will contain a count of the tasks currently sharing this DDS.

The locks that will be defined during initialization are DD lock/share ECB chains just explained. All functions which modify those chains (DDSLOCK, DDSUNLOCK, DDSHARE, DDSUNSHARE) must first acquire a lock on the DDS lock/share chain in question.

A sample JCL deck to run a Special Real Time Operating System test program using DDS follows:

```
//SRTOS         EXEC    PGM=DPPINIT                     Card 1
//STEPLIB       DD      DSN=EMS370.LM71,DISP=SHR        Card 2
//DBINIT        DD      DSN=EMS370.DB171,DISP=SHR       Card 3
//DBINIT2       DD      DSN=EMS370,DB271,DISP=SHR       Card 4
//MSGDS         DS      DSN=EMS370,DB471,DISP=SHR       Card 5
//DDSTATUS      DD      DSN=EMS370,DDS71,DISP=SHR       Card 6
//DDSEQ         DD      DSN=DDS1,DISP=SHR               Card 7
//DDSEQ1        DD      DSN=DDS2,DISP=SHR               Card 8
//DDBPM1        DD      DSN=DDSBP1,DISP=SHR             Card 9
//DDBPM2        DD      DSN=DDSBP2,DISP=SHR             Card 10
//DDSCMPIN      DD      UNIT=DISK,DISP=(,PASS),
                        SPACE=(TRK,(1,1))               Card 11
//MSGOUT        DD      SYSOUT=A                        Card 12
//SYSPRINT      DD      SYSOUT=A                        Card 13
//SYSUDUMP      DD      SYSOUT=A                        Card 14
//COMPRINT      DD      SYSOUT=A                        Card 15
//DDSCTLIN      DD      *                               Card 16
                DDSNAMES (DDSEQ,DDSEQ1)                 Card 17
DDSBPAM         DDSNAMES (DDBPM1,DDBPM2=OUT)            Card 18
//SYSINIT       DD      *                               Card 19
                TCB     1                               Card 20
TO              PATCH   EP=TESTPGM1,TASK=TESTPGM1       Card 21
                WAIT    TO                              Card 22
                ABEND   1                               Card 23
/*                                                      Card 24
```

Card 1 — The entry point for a Special Real Time Operating System execution is DPPINIT.

Card 2 — The Special Real Time Operating System load modules exist on this data set in executable form.

Cards 3-4 — These data sets contain the required arrays for Special Real Time Operating System data base.

Card 5 — This data set contains the messages previously created during Special Real Time Operating System SYSGEN.

Card 6 — The data set will contain a copy of the DDS control areas to keep a current status of all DDS declarations.

Cards 7-10 — These data sets will be the two duplicate data set pairs for this test run.

Card 11 — This data set is a one-track sequential data set which will be used by DDS if a DDS COMPARE function is requested.

Card 12 — This data set will receive Special Real Time Operating System output messages.

Card 13 — This data set will receive initialization messages output.

Card 14 — This data set will receive a dump if one should occur.

Card 15 — This data set will receive the output of IEBCOMPR if a DDS Compare Function is requested.

Card 16 This data set contains the DDS input cards.

Card 17 This card declares that DDSEQ is a duplicate data set name, that DDSEQ is the primary DDNAME, that DDSEQ1 is the backup and that the backup is in-service.

Card 18 This card declares that DDSBPAM is a duplicate data set name, that DDBPM1 is the primary DDname, that DDBPM2 is the backup DDname, and that the backup is out-of-service.

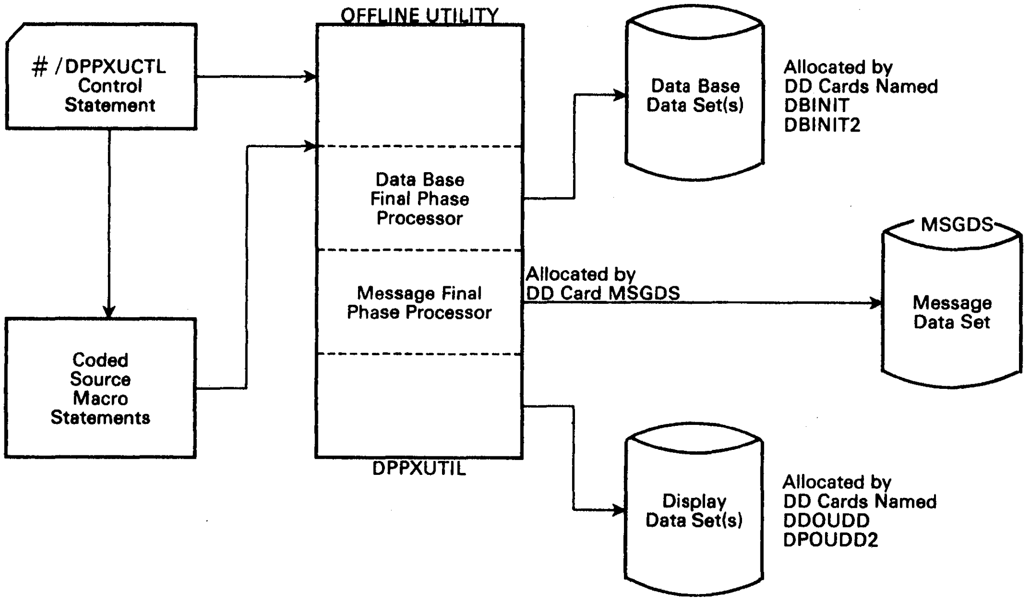
Card 19 This data set contains the Special Real Time Operating System initialization input cards.

Cards 20-23 These cards control the Special Real Time Operating System execution.

Introduction

A realtime system typically requires a detailed description of the environment in which it operates. This description contains information of two types. The first is the selection of options that are to be This includes both hardware and software options, which are selected at installation or system generation (SYSGEN) time. The second type of environment description encompasses those parameters that are of a more dynamic nature. In the Special Real Time Operating System, these consist of display, data base, and message definitions. These definitions are initially made at Special Real Time Operating System SYSGEN time through the use of the offline utility program DPPXUTIL. This same program (DPPXUTIL) is used, as the realtime system develops, to add new definitions or to change old ones. The offline utility program may be run in a partition during an online run, or on a backup CPU.

The data base and message data sets are created and updated using the offline utility. The control cards and macro statements coded by the user result in the data sets being created to the user specification. This is shown in Figure 3-7.



# /DPPXUCTL Control Statement
Coded Source Macro Statements
OFFLINE UTILITY
Data Base Final Phase Processor
Message Final Phase Processor
DPPXUTIL
Allocated by DD Card MSGDS
Data Base Data Set(s)
Allocated by DD Cards Named DBINIT DBINIT2
MSGDS
Message Data Set
Display Data Set(s)
Allocated by DD Cards Named DDOUDD DPOUDD2

Figure 3-7. Offline Utility Processing Overveiw

The control statement (#/ DPPXUCTL) defines to the offline utility data set(s) that is to be created or modified, the locations of the source macro statements to be used to create or modify the data set, and the function to be performed on the data set, i.e., ADD, DEL, REPL, or TEST. The format of the required source macro statements is different for each of the three types of output data sets, and the description of these macros follows in the final phase processor descriptions.

In addition, a facility is provided by the offline utility to allow the user to modify his source macro statement data set prior to its use in generating the output data set. The user requests the update function with a #/ DPPXUPDT control card. The offline utility program invokes the OS/VS1 update utility program IEBUPDTE. Therefore, the user can code IEBUPDTE statements, pass them to DPPXUTIL, and have his source macro data set updated in the same execution as the creation of the output data sets. It should be noted that the offline utility processes in the same sequence as the control statements it receives. As a result, if the update is to take place before the online data set is modified or created, the DPPXUPDT card must precede the DPPXUCTL statement in the input stream. No limit is imposed by the offline utility on the number of control statements that may be used in one execution.

Input to the offline utility must be either cards or blocked or unblocked card image records from a source library. The input consists of control statements, which define the operation to be performed by the offline utility and source macro statements. The macro statements may be in the input stream or in a source library. The macro source library cannot contain control cards. The control statement may be in the input stream or may be a sequential data set.

Each control statement consists of an identifier, an operation, and parameters. The identifier consists of the characters #/ in columns 1 and 2 to denote a control card. The operation must be preceded and followed by at least one blank. The operation describes the function to be performed by the offline utility. DPPXUCTL specifies that an online data set is to be created or modified. DPPXUPDT specifies that a source macro data set is to be updated. The parameters further The parameters further describe the operation to the utility.

The utility program begins processing by looking in the SYSIN data set for a control statement. The first statement should be a control statement; if it is not, an error message is issued, and data in SYSIN will be bypassed until a control statement is found. When a control statement is found, it is validity checked, any errors will cause error messages to be issued and a search begins for the next control statement. Control statement errors do not terminate the utility program; however, processing for the control statement in error will be bypassed with appropriate diagnostic messages. The offline utility program terminates when no more control statements are to be processed.

When a valid DPPXUCTL control statement has been processed, all the input source macro statements for the control statement are read in and rewritten into the data set allocated to the job by the SYSUT4 DD card. This data set is then passed to the assembler. When a valid DPPXUPDT control statement is processed, the IEBUPDTE control and data statements (which must follow the DPPXUPDT statement in the SYSIN input stream) are read in and rewritten to the SYSUT4 data set. This data set is then passed to IEBUPDTE.

Source Data Set Update Operation

The source macro statements used to define an online data set may be maintained as a sequential data set or as a member of a partitioned data set. These source macro data sets may be created and modified by the offline utility. The modification of a source macro data set is invoked by the #/ DPPXUPDT control statement. Figure 3-8 shows an overview of the update processing.

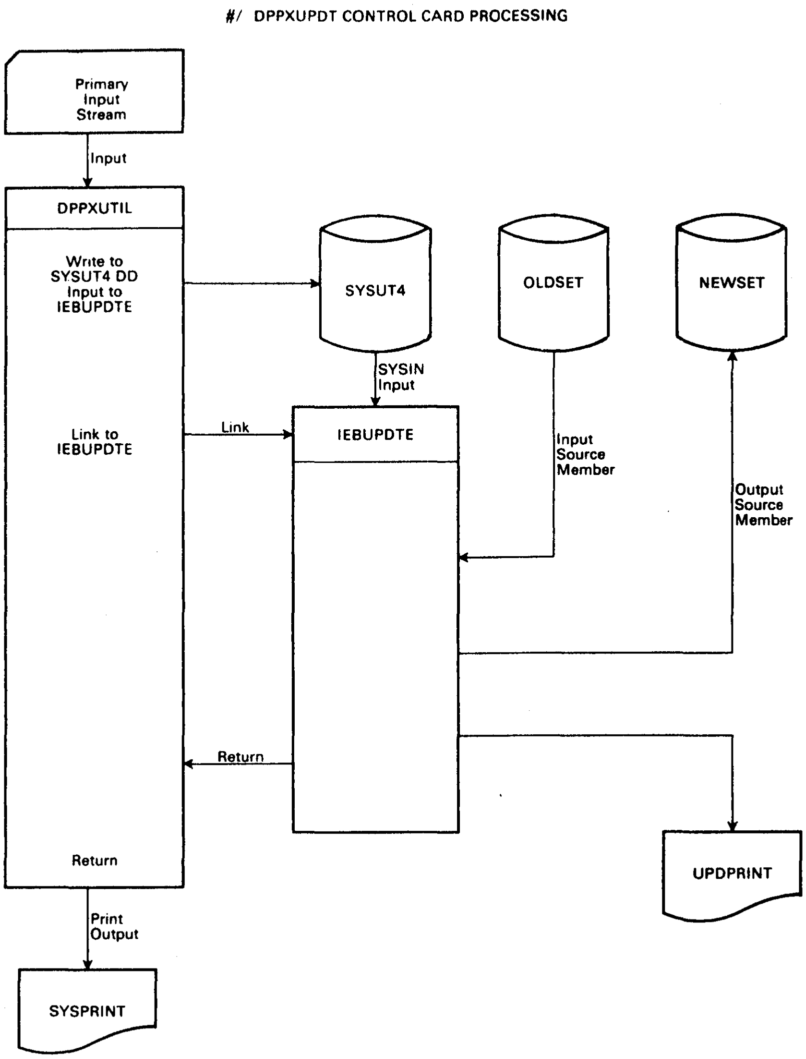


Figure 3-8. Update Processing Overview

The format of the DPPXUPDT control statement is:

```
  #/   DPPXUPDT      OLDSET=ddname,NEWSET=ddname

  #/   Is required in columns 1 and 2.
```

DPPXUPDT
Specifies a source data set update and must be preceded and followed by at least one blank.

OLDSET=ddname
Specifies the ddname of the DD card allocating the data set to be used as input (SYSUT1) by IEBUPDTE.

NEWSET=ddname
Specifies the ddname of the DD card allocating the data set to be used as output (SYSUT2) by IEBUPDTE.

The IEBUPDTE control statements and input data statements must be coded as specified in the *OS/VS1 Utilities Manual*, GC35-0005, and must immediately follow the DPPXUPDT control statement in the SYSIN input stream. Standard assembler language rules apply to comments and continuation.

The following example shows a typical offline utility input stream for an update function.

```
#/      DPPXUPDT    OLDSET=DBASIN,NEWSET=DBASOUT
./      ADD       NAME=DBAS1,LIST=ALL
        ARRAY          NAME=ARRAY01
                  ITEM NAME=ITEM101,TYPE=F
                  ITEM NAME=ITEM102,TYPE=F
        ARRAY          NAME=ARRAY02
                  ITEM NAME=ITEM201,TYPE=C,LEN=16
./      CHANGE         NAME=DBAS2,LIST=ALL
./      DELETE         SEQ1=200,SEQ2=300
                  ITEM NAME=ITEM705,TYPE=F    00000500
./      REPL       NAME=DBAS3,LIST=ALL
        ARRAY          NAME=ARRAY05
                  ITEM NAME=ITEM501,TYPE=A
```

Source Data Set Update Control Card Example

In this example, DD cards named DBASIN and DBASOUT may define the same or different data sets. A new member named DBAS1 will be created in the data set defined by DD statement DBASOUT. Existing member DBAS2 will have statement numbers 200 through 300 deleted, and statement number 500 will be replaced. Existing member DBAS3 will be replaced.

Processing an Online Data Set

The DPPXUCTL control statement requests the creation or modification of a data set which is normally used online. The utility program, after reading the source macro data set and rewriting it to the SYSUT4 data set, links to the assembler. The link will be to the OS/VS1 assembler unless the user requests the use of the assembler H program product (5734-AS1) by coding PARM=H on the JCL EXEC card. The data in SYSUT4 is assembled and control is returned to the offline utility. If the assembler return code is 8 or greater, processing for this control statement is aborted, and the utility attempts to read another control card. If the return code is less than 8, the utility loads the OS/VS1 Loader, which loads the assembled module into virtual storage. Control is returned to the utility and if the return code is less than 8, the appropriate final phase processor is invoked by a link to update the online data set. The appropriate final phase processor depends upon which operand was coded in the AREA= keyword of the control statement. Figure 3-9 shows an overview of the online data set processing function.

The format of the DPPXUCTL control statement is shown below. Standard assembler language rules apply to comments and continuation with the exception that each parameter must not be split across two cards; that is, each parameter must be wholly contained on one card.

```
#/    DPPXUCTL    AREA= { DISPDEF }  ,INPUT= { *                   }
                        { DBDEF   }          { ddname              }
                        { MSGDEF  }          { ddname(membername)  }

                  [         { ADD  } ]
                  [ ,OPTION={ DEL  } ] [,TYPE=device type]
                  [         { REPL } ]
                  [         { TEST } ]
```

#/ Must be in columns 1 and 2.

DPPXUCTL
Specifies an online data set is to be modified or created. This must be preceded and followed by at least one blank.

AREA=
Must be specified and must specify one of three keywords:

DISPDEF
Specifies that the operation be performed against the display data set.

DBDEF
Specifies that the operation be performed against the data base data set.

MSGDEF
Specifies that the operation be performed against the message data set.

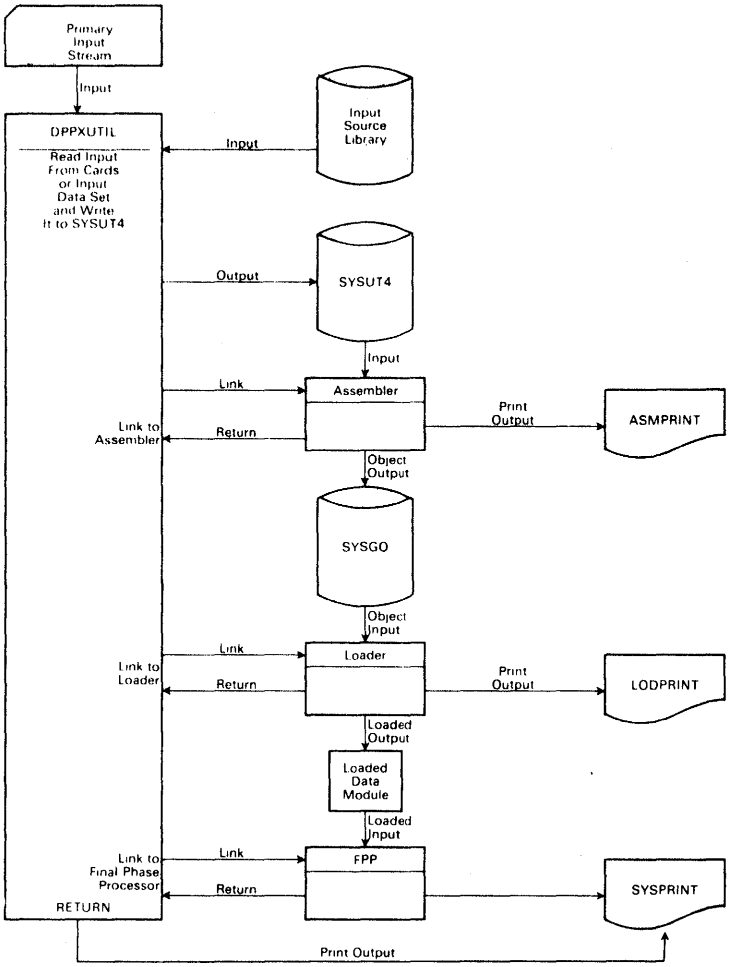


Figure 3-9. Online Data Set Processing Overview

INPUT=
Must be specified and must specify one of the following:

*
Specifies that the input for this execution immediately follows the control statement in the.SYSIN input stream.

ddname
Specifies that the input for this execution is in the sequential data set allocated to the job by the named (ddname) DD statement.

ddname(member name)
Specifies that the input for this execution is in the "member name" member of the partitioned data set allocated to the job by the named (ddname) DD statement. The ddname and member name may consist of from 1 to 8 alphabetic (A-Z) or numeric (0-9) characters, the first of which must be alphabetic. Special characters @, #, and $ are not allowed.

OPTION=
Must be specified; it indicates the type of operation to be performed. One of the following must be specified:

ADD
Add a new member to the online data set.

REPL
Indicates to replace an existing member in the online data set and if the member does not exist, to add the new one.

DEL
Signifies to delete an existing member from the online data set.

TEST
Is similar to REPL; the member is assembled, listing produced and so forth, but the content of the online data set is not changed.

TYPE=
Is optional and is recognized only when AREA=DISPDEF is specified. If AREA=MSGDEF or DBDEF,TYPE= is ignored. When coded, it must specify the device type of the display hardware, i.e., 3277-1, 3277-2, 5985. The default, if AREA=DISPDEF and TYPE= is not coded, is 3277-2.

The following example shows a typical input stream to the offline utility to process an online data set.

```
#/ DPPXUCTL     AREA=DBDEF,INPUT=*,OPTION=ADD
               ARRAY NAME=ARRAY09
                   ITEM NAME=ITEM901,TYPE=F
                   ITEM NAME=ITEM902,TYPE=F
#/ DPPXUCTL     AREA=DBDEF,OPTION=REPL,                   *
                INPUT=DBASOUT(DBAS1)
#/ DPPXUCTL     AREA=MSGDEF,OPTION=REPL                   *
                INPUT=MSGIN(TIMEMSG)
#/ DPPXUCTL     AREA=MSGDEF,OPTION=DEL,                   *
                INPUT=MSGSEQ
```

Online Data Set Update Example

In the preceding example, the following processing is being requested.

1. A new member (ARRAY09) is being ADDed to the online data base data set. The member will be created from the ARRAY and ITEM cards following the control statement in the input stream (INPUT=*).

2. The member named DBAS1 from the source macro input data set allocated by DD card DBASOUT is to be assembled. The resulting member(s) will REPLace corresponding members in the online data base data set.

3. The member named TIMEMSG from the source macro input data set allocated to the job by the DD card named MSGIN is to be processed. The resulting member(s) will REPLace corresponding members in the online message data set.

4. The sequential source macro input data set allocated to the job by the DD statement named MSGSEQ is to be processed. The resulting member names will be DELeted from the online message data set.

The following example is typical of the JCL required to execute the offline utility program (DPPXUTIL). Following is a description of each of the JCL statements in the example. The underlined portions of the JCL will likely have to be changed by the user to suit the requirements of his operation.

```
//BUILD         JOB     (ACCOUNTING INFORMATION)
//S1            EXEC    PGM=DPPXUTIL,PARM=H
//STEPLIB       DD      DSN=USER.LINKLIB,DISP=SHR
//SYSPRINT      DD      SYSOUT=A
//ASMPRINT      DD      SYSOUT=A
//UPDPRINT      DD      SYSOUT=A
//LODPRINT      DD      SYSOUT=A
//SYSLIB        DD      DSN=USER.MACLIB,DISP=SHR
//              DD      DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1        DD      UNIT=(SYSDA,SEP=SYSLIB),SPACE=(CYL,(2,2))
//SYSUT2*       DD      UNIT=(SYSDA,SEP=SYSUT1),SPACE=(CYL,(2,2))
//SYSUT3*       DD      UNIT=(SYSDA,SEP=SYSUT1),SPACE=(CYL,(2,2))
//SYSUT4        DD      UNIT=(SYSDA,SEP=SYSUT1),SPACE=(CYL,(2,2))
//  DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//DBINIT        DD      DSN=USER.DB1,DISP=OLD
//DBINIT2       DD      DSN=USER.DB2,DISP=(MOD,PASS),DCB=(DSORG=DA)
//MSGDS         DD      DSN=USER.MSG,DISP=OLD
//DPOUDD        DD      DSN=USER.DISP1,DISP=OLD
//DPOUDD2       DD      DSN=USER.DISP2,DISP=OLD
//DBASIN        DD      DSN=USER.SOURCE.DB.MACROS,DISP=OLD
//DBASOUT       DD      DSN=USER.SOURCE.DB1.MACROS,DISP=OLD
//MSGIN         DD      DSN=USER.SOURCE.MSG.MACROS,DISP=OLD
//MSGSEQ        DD      DSN=USER.SOURCE.SEQMSG.MACROS,DISP=OLD
//SYSGO         DD      UNIT=SYSDA,SPACE=(CYL,(1,1)),
//  DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSIN         DD      *
/*
```

(Input Control Statements)

JCL Example

*Not required when "PARM=H" is specified on the execute card.

JOB
Is a standard OS/VS1 job card; the accounting information is dependent upon individual installation requirements.

EXEC
Is a standard OS/VS1 EXEC card; it must specify PGM=DPPXUTIL or an applicable user PROC.

PARM
The offline utility will provide the option to print or not to print statements generated by the processing of a macro. This will be accomplished by the offline utility inserting or not inserting a PRINT NOGEN statement as the first statement in the Assembler SYSIN stream. Control will be provided through the PARM keyword operand on the execute card for DPPXUTIL. This option is provided in addition to the option to select the OS/VS1 assembler or the H assembler.

The following values may be specified:

F	--	Selects the OS/VS1 Assembler.
H	--	Selects the H Assembler.
GEN	--	Print macro generated statements.
NOGEN	--	Do not print macro generated statements.

In all cases, the default values will be "F" and "NOGEN".

Valid combinations of the values are:

```
 PARM = 'F'
 PARM = 'H'
 PARM = 'GEN'
 PARM = 'NOGEN'
 PARM = 'F,GEN'
 PARM = 'F,NOGEN'
 PARM = 'H,GEN'
 PARM = 'H,NOGEN'
```

If an invalid value is specified for the PARM operand or if the PARM operand is omitted, the default of PARM='F,NOGEN' will be used and message DPPXUT25 will be printed.

STEPLIB DD

Defines the library containing the DPPXUTIL program and final phase processors and is not required if these programs reside in SYS1,LINKLIB.

SYSPRINT DD

Defines a data set in which printed output will be placed, or may specify a standard output class.

ASMPRINT DD

(Same as SYSPRINT) for printed output from the assembler.

UPDPRINT DD

(Same as SYSPRINT) for printed output from IEBUPDTE.

LODPRINT DD

(Same as SYSPRINT) for printed output from the loader. This may be a DD DUMMY to reduce printed output.

SYSLIB DD

Defines the data set(s) containing the macros used by the assembler.

SYSUT1 DD

Defines the assembler work data sets. The device classname SYSDA defines a direct-access device. This name (SYSDA), if used, must have been generated into the OS/VS1 system. SEP= is specified to improve assembler performance.

SYSUT2 DD

(Same as SYSUT1). Not required when "PARM=H" is specified on the execute card.

SYSUT3 DD

(Same as SYSUT2).

SYSUT4 DD

Defines a work data set for DPPXUTIL. The DCB parameters must specify RECFM=FB and a BLKSIZE that is a multiple of 80. The LRECL must be 80.

DBINIT DD

Defines the data base partitioned data set that contains a member for every array in the data base, control information for direct access resident arrays and initial data for VS resident arrays. This DD card is required if any utility control card specifies AREA=DBDEF.

DBINIT2 DD

Defines the BDAM data set which contains the initial data for DA resident arrays. This DD card is required if any utility control statement specifies AREA=DBDEF. The data set described by this DD card must be allocated prior to the execution of the offline utility. The DISP= operand on the DD card must specify (MOD,PASS).

MSGDS DD
Defines the data set containing online display information. This DD card is required if any utility control statement specifies AREA=MSGDEF.

DPOUDD DD
Defines the data set containing online display information. This DD card is required if any utility control statement specifies AREA=DISPDEF.

DPOUDD2 DD
Defines the display online comment data set. This DD card is required if any utility control statement specifies AREA=DISPDEF and the DISPGEN statement specifies COMMENT=YES.

DBASIN DD
May have any DD NAME. In this example, the DBASIN DD name was used to correspond to the OLDSET= name in the source data set update control card example. The name chosen on the OLDSET= may be any valid DD name; however, it must have a corresponding DD statement.

DBASOUT DD
(Same as DBASIN) for NEWSET= keyword

MSGIN DD
(Same as DBASIN) for INPUT= in the online data set update example

MSGSEQ DD
(Same as DBASIN) for INPUT= in the source data set update control card example

SYSGO DD
Defines the data set to contain the object deck output from the assembler. This data set is used as input to the OS/VS1 loader.

SYSIN DD
Defines the input from which DPPXUTIL gets its control statements as possibly some source macro statements.

Message Final Phase Processor

The message final phase processor accepts the load modules created as a result of offline utility processing of DEFMSG statements and puts them into the online message data set. Each DEFMSG statement results in a member being processed in the partitioned data set allocated by the MSGDS DD card.

The type of processing is determined from the request in the control card, e.g., ADD, DEL, REPL, or TEST.

The following is an example of offline utility control statements that would result in the invoking of the message final phase processor.

```
#/ DPPXUCTL AREA=MSGDEF,INPUT=*,OPTION=ADD
   DEFMSG 7,ROUTE=200,ACT=I,TEXT='DUMMY MESSAGE'
```

A control statement such as this would cause message number 7 to be added to the online message data set. The message would be a member with the name DPP007.

There are more examples and additional descriptions of the DEFMSG in this manual in the section describing the online message handler.

Data Base Final Phase Processor

The data base final phase processor receives control from the offline utility to process the assembled and loaded input created by the input cards following the AREA=DBDEF card. The function of the data base final phase processor is to build and modify the online data base data sets.

The input is the assembled and loaded data generated from the user coded ARRAY, BLOCK, and ITEM macros. These macros are used offline only and are described in detail in the following section.

For the purpose of the following discussion, an ARRAY is defined as an arrangement of data ITEMS in one or more dimensions or BLOCKS. The Special Real Time Operating System arrays with data items of one dimension only are called UNBLOCKED arrays. Arrays with two or more dimensions are BLOCKED arrays.

Arrays which will reside in virtual storage during online processing are known as VS resident arrays. A VS resident array may be either a BLOCKED array or an UNBLOCKED array. An array which resides on a direct access device during online execution is known as a DA resident array. A DA resident array must be a BLOCKED array.

A 'LOGGABLE' array is a VS array for which logging has been requested. A 'LOGGING' array is a DA resident array into which a VS resident logable array is being logged. For a more detailed description of data base logging refer to the section in Chapter 2 entitled, "Data Base Logging."

The Special Real Time Operating System data base consists of VS resident arrays and DA resident arrays. Data base logging will be performed on a demand basis or on a cyclic basis if cyclic logging is SYSGENed.

The offline utility program and the data base final phase processor create and update the data base. The type of array and the operation to be performed are defined through the utility control statements and the offline macros ARRAY, ITEM, and BLOCK.

The ARRAY macro is used to define the array, its characteristics, and its dimensions. The BLOCK macros define the boundaries within the dimensions of the array. The ITEM macro defines each item or element of the array and its initial values. An item control block is created to define each ITEM defined. The item control block contains the item name, the type of data, the length of data, the repetition factor of the item, and the displacement into the array of the start of the data item.

The operation to be performed on the data base is defined to the offline utility by the OPTION= parameter on the #/ DPPXUCTL input control statement. The operation types and meanings are shown as follows:

ADD
Indicates a new array is to be added to the data base. If the data base already contains an array with the same name, the new array will not be added, and an error message will be issued.

REPL
Indicates that a new array is to be added to the data base. If the data base does not contain an array with the same name, the array will

be added. If the data base does contain the named array, it will be replaced in the data base.

DEL
Indicates that an existing array is to be deleted from the data base. If the array does not exist, an error message is written.

TEST
Is similar to REPL except the data base is not modified.

Data Base Data Sets

The Special Real Time Operating System data base consists of one partitioned data set (DSORG=PO) and one or more direct data sets (DSORG=DA). The partitioned data set (PDS) is allocated at the Special Real Time Operating System SYSGEN time and is referenced in realtime by the DD card named DBINIT. The direct data set is also allocated at the Special Real Time Operating System SYSGEN time and is referenced by the //DBINIT2 DD card.

The PDS contains a directory entry for every array in the data base. Information in the directory entry is used for data base initialization. The members of the PDS contain the Item Control Blocks for each array in the data base. For VS resident arrays the member containing the Item Control Block also contains the VS resident array and its initial data. For DA resident arrays the PDS member containing the Item Control Block also contains control information used to locate the corresponding DA array in the direct data set.

There can be only one PDS in the data base. However, additional direct data sets can be allocated and become part of the data base. Once an additional direct data set has been referenced during execution of the offline utility, any further reference to this data set as part of the data base must be made through a DD card with the DD name the same as the DD name used on the DADD or the LOGDD keyword on the ARRAY macro used to create the array. This is because the DD name becomes part of the control information written into the PDS member for the array.

Secondary data bases may be created by the user. To do this, he would create a PDS and one or more direct data sets and reference them through the DBINIT and DBINIT2 DD cards.

Warning: The data base data sets contain records with references to and dependencies on other records and members. These references and dependencies are constructed by the offline utility program DPPXUTIL, and the data sets must not be modified except by DPPXUTIL. This precludes moving or copying an entire data base data set, and also prohibits modifying their content by adding or deleting records or members, or by changing block sizes. Concatenation of data base data set groups for realtime execution is not allowed.

OFFLINE MACROS

The following pages describe the operands and functions of the offline macros. For convenience they are listed in alphabetical order.

Some of the operands on the data base offline macros have been designed to accept self-defining terms or absolute expressions in place of an actual value. This will provide greater flexibility in design and maintenance of data base arrays. The macros and operands which provide this capability are shown below.

<u>MACROS</u>	<u>OPERANDS</u>
ARRAY	BLKCT= BLKSIZE=
BLOCK	start number ,stop number
ITEM	LEN= RPT= DISP=

EXAMPLE:

The following will illustrate some self-defining terms and absolute expressions, the format of that can be used in the ARRAY, BLOCK, and ITEM macros.

```
DATA          DSECT
COUNT          EQU     10
START          DS      0D
A              DS      F
B              DS      D
C              DS      F
STOP           DS      0D
SIZE           EQU     STOP - START
               ARRAY   NAME=ARAY,BLKCT=(COUNT),BLKSIZE=(SIZE)
               BLOCK  (L'A),(C-B)
               ITEM TYPE=C,LEN=(L'A),DISP=(A-START)
               ITEM TYPE=C,LEN=(COUNT),DISP=4,RPT=(L'B)
```

Note:

- Self-defining terms and absolute expressions must be enclosed in parenthesis.
- Care must be taken not to become "H" assembler dependent.
- No validity checking of block numbers will be done in the BLOCK macro if a self-defining term or absolute expression is used on a block macro.
- No validity checking will be done to prevent items from overlapping when the "DISP=" operand is used.

ARRAY

The ARRAY macro is used to define a data base array to the database offline utility.

```
[symbol]   ARRAY    {NAME=name     }
                    {NUMBER=number }

                    [,INIT=     {NO }]
                    [           {YES}]

                    [,REINIT=   {NO }]
                    [           {YES}]

                    [,LOCATE=   {VS}]
                    [           {DA}]

                    [           {BLOCK                   }]
                    [,BLKCT=    {number                  }]
                    [           {(self-defining term)    }]
                    [           {(absolute expression)   }]

                    [           {number                  }]
                    [,BLKSIZE=  {(self-defining term)    }]
                    [           {(absolute expression)   }]

                    [,DADD= ddname]

                    [,USE=      {  7  }]
                    [           {value}]

                    [           {DBLWD}]
                    [,BNDRY=    {PAGE }]
                    [           {MIN  }]

                    [LOGNAME= name]

                    [LOGDD= ddname]

                    [LOGFREQ=   {  0  }]
                    [           {value}]

                    [LOGCOPY=   {  1  }]
                    [           {value}]

                    [LOGWRAP=name]
```

NAME=
Is a 1 to 8 byte alphameric name that conforms to standard OS naming conventions for members of a partitioned data set. The array name for each array must be unique for all arrays in the data base. The NAME and NUMBER parameters are mutually exclusive.

NUMBER=
Is a decimal number from 1 through 255 by which the array may be referenced during online data base processing. The total number of arrays in the data base is not limited to 255, however, the maximum number of NUMBERed arrays is 255. An entry will be allocated in the online tables for each number from one to the highest assigned array number even though all numbers do not have an array built for them (i.e., two arrays are created - NUMBER=2 and NUMBER=10 - this will create only two arrays in the data base but will create 10 entries in the online tables.) For this reason, numbers should be assigned in ascending sequence starting with one. Skipped numbers are valid, but will result in wasted virtual storage and extra processing time for online data base processing. The NAME and NUMBER parameters are mutually exclusive.

INIT=
Is used to determine whether a VS resident array is to be initialized at data base initialization time. If YES is specified, space will be allocated in VS, and the data specified in the ITEM cards for this array will be moved from a direct access device into the allocated space in VS. If NO is specified, space will be allocated in VS, and no data will be moved into the allocated space. (The space may contain residue data from previous programs.) The The default value for this parameter is NO. This parameter is ignored if DA is specified on the LOCATE parameter.

REINIT=
Defines reinitialization action to be taken after a Special Real Time Operating System restart. The parameter is valid only if LOCATE=VS is specified and if logging is specified for the array. If REINIT=YES is specified, the data from the most recently logged copy of the array will be read into the space allocated to the array after the restart occurs. Reinitialization will be bypassed by the data base online initialization routines if the logged copy is not at the same update level as the array which was loaded at the Special Real Time Operating System initialization. This would occur if the array had been redefined through the offline utility programs.

If REINIT=NO is specified or the parameter is omitted, the content of the array will not be modified after a restart occurs.

LOCATE=
Is used to determine whether the array is to reside on a direct access device or in virtual storage during online data base processing. If VS is specified, the array will be initialized in virtual storage in accordance with specifications of the INIT parameter. If DA is specified, no initialization will take place at data base initialization time. The default value for the LOCATE parameter is VS.

BLKCT=
Determines whether the array is blocked or unblocked. If this parameter is omitted, the array is assumed to be unblocked and, therefore, must reside in virtual storage. A number from 1 through 32767 will specify the exact number of data blocks that will be created for this array. The number of data blocks can be implied by specifying BLOCK on the BLKCT This indicates that the highest block number specified on a BLOCK macro for this array will determine the number of data blocks for this array. BLKCT is required if DA is specified

on the LOCATE parameter. When BLKCT=n is specified, there cannot be either more BLOCK macros than n or a BLOCK macro cannot specify a number greater than n.

Examples:

```
      ARRAY           NAME=EXAMP1,BLKCT=2
          BLOCK
                      ITEM
          BLOCK
                      ITEM
          BLOCK
                      ITEM
```

This is invalid because BLKCT=2 is specified but the array has 3 BLOCK macros.

```
      ARRAY           NAME=EXAMP2,BLKCT=4
          BLOCK       1,5
                      ITEM
```

This is invalid because BLKCT=4 was specified and the BLOCK macro has a request for 5 blocks.

```
      ARRAY           NAME=EXAMP3,BLKCT=10
          BLOCK       2,5
                      ITEM
```

This example is valid and will build an array with 10 blocks in which blocks 2 through 5 will be created with the initial data as requested in ITEM cards and blocks 1 and 6 through 10 will also be created but will have initial data of binary zeros.

If BLKCT=BLOCK were specified, the number of blocks created would be the same as the highest block number generated by a BLOCK macro.

```
      ARRAY           NAME=EXAMP4,BLKCT=BLOCK
          BLOCK       1,5
                      ITEM
          BLOCK
                      ITEM
          BLOCK
                      ITEM
```

The above example would result in a block count of 7 as block numbers are assigned sequentially when no operands are specified.

BLKSIZE=
Specifies the number of bytes to be allocated to each block of data in this array. This parameter is ignored if the BLKCT parameter is omitted. If the BLKSIZE parameter is omitted and the BLKCT parameter is specified, the size of the first data block described by ITEM macros for this array will determine the block size for all blocks in this array. The maximum block size is limited to the track capacity of the device to which the array will be allocated. If the amount of data specified in the ITEM cards for the first BLOCK is greater than the data set block size, the data block will be truncated to data set block size and this will be the size for all subsequent blocks. If subsequent blocks generate less data than the first BLOCK, they will be padded with binary zeros to the same size as the first block. If subsequent blocks exceed the size of the first block, they will be truncated. When BLKSIZE=n is specified, each BLOCK will be created n bytes long.

DADD=
Specifies the name of a data definition (DD) statement which describes a BDAM data set where space for this direct access resident array will be allocated. This parameter is required if DA is specified on the LOCATE parameter; however, if VS is specified on the LOCATE parameter, the DADD parameter is ignored.

USE=
Is a code from 1 to 7 which indicates the expected frequency of reference to items in the array. The arrays are loaded into virtual memory based upon this code. Code 1 indicates the highest usage, and code 7 has the lowest usage. If the USE parameter is omitted or if an invalid value is specified, a value of 7 will be assumed as the default use code. This parameter is ignored if DA is specified on the LOCATE parameter.

BNDRY=
Is used to determine the boundary alignment for a virtual storage resident array at data base initialization time. If the parameter is omitted or if DBLWORD is specified, the array may be initialized starting on any doubleword boundary. If PAGE is specified, the array will be initialized to start on a virtual storage page boundary. Specification of MIN will cause Data Base Initialization to do calculations based on the array length and position the start of the array so that it will be contained in the smallest possible number of virtual storage pages.

Those arrays for which logging is specified will have a 24-byte logging header appended to the front of the space allocated in VS. For boundary alignment purposes, the first byte of the logging header will be considered the start of the array.

The following operands describe the logging array associated with a VS array and the logging characteristics of the array being defined. Logging of the array will not be allowed if they are not specified. Since a DA resident array is allocated in response to these operands, they should not be specified if logging is not required.

LOGNAME=
Specifies a 1 to 8 character name to be used as the array name of a direct access resident array where the virtual storage resident array is to be logged. The log array name must conform to the same standards and conventions as set forth under the NAME parameter. This parameter is ignored if DA is specified on the LOCATE parameter. If this parameter is omitted and VS is specified on the LOCATE parameter, no logging will be performed.

LOGDD=
Specifies the name of a data definition statement which describes a BDAM data set where space can be allocated for the logging array where this array is to be logged. This parameter is required if VS is specified on the LOCATE parameter and if a name was specified on the LOGNAME parameter. This parameter is ignored if DA is specified on the LOCATE parameter.

LOGFREQ=
Indicates by a code of 0 to 3 the frequency at which this array is to be logged. A code of 0 indicates that it is to be logged only on demand. Codes 1 to 3 are used in conjuntion with system generation parameters to specify the log frequency. A code of 1 is the highest frequency, and 3 is the lowest frequency. If the LOGFREQ parameter is omitted, or if an invalid value is specified, a value of 0 will be assumed. This parameter is ignored if DA is specified on the LOCATE parameter.

LOGCOPY=

Specifies the number of history copies of this array for which space is to be allocated in the logging array. If the LOGCOPY parameter is omitted or if 0 is specified, a value of 1 will be assumed as the default value. This parameter is ignored if DA is specified on the LOCATE parameter.

LOGWRAP=

Specifies the name of a user-written load module to be given control when the last block of the logging array has been filled and wrap around will occur on the next request to log this array. This parameter is ignored if DA is specified on the LOCATE parameter.

The load module will be entered via a PATCH to the load module as a dependent task. The parameter field will be eight bytes and contains the name of the array for which the logging array wrapped around.

The following chart shows the ARRAY macro operands, which are required and which are optional for the creation of any type array.

Operands	VS Array Unblocked	VS Array Blocked	DA Array Blocked	VS Array with Logging Unblocked	VS Array with Logging Blocked
NAME NUMBER	R	R	R	R	R
INIT	O	O	I	O	O
REINIT	O	O	I	O	O
LOCATE	O	O	R	O	O
BLKCT	*	R	R	*	R
BLKSIZE	I	O	O	I	O
DADD	I	I	R	I	I
USE	O	O	I	O	O
BNDRY	O	O	I	O	O
LOGNAME	**	**	I	R	R
LOGDD	I	I	I	R	R
LOGFREQ	I	I	I	O	O
LOGCOPY	I	I	I	O	O
LOGWRAP	I	I	I	O	O

R - REQUIRED OPERAND
O - OPTIONAL OPERAND
I - IGNORED OPERAND
* - THE PRESENCE OF THIS OPERAND WOULD CHANGE THE ARRAY FROM UNBLOCKED TO BLOCKED
** - THE PRESENCE OF THIS OPERAND WOULD CHANGE THE ARRAY INTO A LOGABLE ARRAY.

BLOCK

The BLOCK macro is used to define data blocks to the Data Base Final Phase Processor. Each block in an array will have identical dimensions. Two consecutive BLOCK macros are not allowed. The BLOCK macro must be followed by an ITEM macro.

symbol	BLOCK	[{ start number (self-defining term) (absolute expression) }]
		[{ end number (self-defining term) (absolute expression) }]

start number
Is the number to be assigned to the data block described by the ITEM macro statements following this macro statement. If this parameter is omitted, the next sequential block number will be assumed. The lowest valid block number is 1.

end number
Is the number to be assigned to the last data block described by the ITEM macro statements following this macro statement. This parameter is not required when only one data block is described by the BLOCK macro. This parameter causes the data block described to be duplicated and assigned a block number for each consecutive number from the start number through the end number. The value of the end number must be greater than the value of the start number.

Block numbers should be assigned in consecutive, ascending sequence starting with the number 1. Missing block numbers between 1 and the highest block number specified will cause the generation of a block of binary zeros to be generated for each of the missing numbers.

If the BLKCT parameter was not specified on the ARRAY macro, the BLOCK macro will be ignored. The BLOCK macro must not specify a block number greater than the value specified in the BLKCT parameter on the ARRAY macro.

EXAMPLES:

```
          ARRAY       NAME=BLKEXAM,BLKCT=BLOCK
          BLOCK
                      ITEM TYPE=H,INIT=21
          BLOCK       10

                      ITEM TYPE=N
```

This example will create an array of 10 blocks with each block two bytes long. Block 1 will be initialized to 21, while 2 through 10 will be binary zeros.

```
      ARRAY           NAME=XMP,BLKCT=10
          BLOCK       3,5
                      ITEM TYPE=F,INIT=-1
```

This example will create an array with ten 4-byte blocks. Blocks 1 and 2 will be initialized to binary zeros, blocks 3, 4, and 5 to binary ones, and blocks 6 through 10 to binary zeros.

```
ARRAY           NAME=BLKEX,BLKCT=BLOCK
    BLOCK       4
                ITEM TYPE=C,INIT='A',LEN=10
    BLOCK       6
                ITEM TYPE=C,INIT='B',LEN=6
    BLOCK       7
                ITEM TYPE=F
    BLOCK       10,15
                ITEM TYPE=F,INIT=-1
    BLOCK
                ITEM TYPE=C,LEN=5
```

The above example will create an array with 16 blocks, each having 10 bytes. Blocks 1, 2, and 3 will each have 10 bytes of binary zeros. Block 4 will be the character 'A' -(HEX 'C1') followed by 9 bytes of blanks (HEX '40'). Block 5 will be 10 bytes of binary zeros. Block 6 will be the character 'B' -(HEX 'C2'), 5 bytes of blanks (HEX '40'), followed by 4 padding bytes of binary zeros. Block 7 will be fullword aligned and will have 4 bytes of zeros (F) followed by 6 bytes of padding, also binary zeros. Blocks 8 and 9 will each have 10 bytes of binary zeros. Blocks 10 through 15 will also be fullword aligned, each containing 4 bytes of binary ones (INIT=-1) followed by 6 bytes of padding (binary zeros). Block 16 will be 10 bytes long with the first 5 bytes being blanks (HEX '40') and 5 bytes of binary zeros.

ITEM

The ITEM macro is used to define, to the data base offline utility, a data item to be contained in a data base array.

```
symbol    ITEM      TYPE = type

                   [ ,NAME= symbol ]

                   [ ,INIT= value  ]

                   [       (number                )]
                   [ ,LEN= {(self-defining term)  }]
                   [       ((absolute expression) )]

                            1
                   [ ,RPT=  (value                )]
                   [       {(self-defining term)  }]
                   [        ((absolute expression))]

                   [        (number               )]
                   [ ,DISP={(self-defining term)  }]
                   [        ((absolute expression))]
```

NAME=
Is a 1 to 8 byte alphanumeric name. The ITEM name for each ITEM must be unique for all items in the entire data base.

ITEM names may be assigned to ITEMs in the first block of a blocked array and will be applied to all blocks of that array. Names may be coded for ITEMs in succeeding blocks, but the names will be ignored; that is, they will not appear in any Special Real Time Operating System ITEM name lists and will not be checked for duplication.

TYPE=
Is required and must be one of the following:

TYPE	DEFAULT LENGTH	DEFAULT ALIGNMENT	DATA TYPE	MAXIMUM LEN=
A	4	Fullword	Address Constant	4
F	4	Fullword	Fixed Point	4
H	2	Halfword	Fixed Point	2
D	8	Doubleword	Floating Point	8
E	4	Fullword	Floating Point	4
P	1	Byte	Packed Decimal	256
B	1	Byte	Binary	256
C	1	Byte	Character	256
X	1	Byte	Hexadecimal	256
N	0	None	None	256

Note: The "A" type must be specified in non-relocatable terms or expressions since it will not relocate addresses.

The "N" type allows the user to give an additional name or 'alias name' to the ITEM which immediately follows the null item, or it allows the user to generate a name which will define one or more of the following items which have no name assigned. The INIT and RPT parameters are ignored.

The following examples shows the use of the TYPE=N operand:

```
ARRAY     NAME=NULLEX
ITEM      NAME=NULL01A,TYPE=N
ITEM      NAME=NULL01B,TYPE=N
ITEM      NAME=NULL01,TYPE=F,INIT=27
```

In this example the null items named NULL01A and NULL01B will have the same displacement into array NULLEX as item NULL01. A GETITEM TYPE=ADDR during online processing for item name NULL01B would return the same address as for name NULL01.

LEN=
May be any integer value from 1 through 256, inclusive. When LEN= is coded, boundary alignment is negated. LEN= coded on a TYPE=N item will determine the length of data to be returned by a GETITEM on a null or alias name. This is shown in the following example:

```
ARRAY     NAME=NULLEX1
ITEM      NAME=NULLALL,LEN=16,TYPE=N
ITEM      NAME=NULL1A,TYPE=A
ITEM      NAME=NULL1B,TYPE=D
ITEM      NAME=NULL1C,TYPE=F
ITEM      NAME=NULLPART,LEN=2,TYPE=N
ITEM      NAME=NULL2C,TYPE=F
```

In this example, an online GETITEM TYPE=DATA for item name NULLALL would get all the data for items NULL1A, NULL1B, and NULL1C. A GETITEM TYPE=DATA for name NULLPART, however, would get only the first two bytes of data from item NULL2C.

The LEN parameter is required for types B and P if initial data is specified on the INIT parameter.

The LEN parameter can be used with type N to determine the total length of subsequent unnamed items.

The LEN parameter for types A, E, and F may be specified as a value from 1 through 4. If the parameter is omitted. or an invalid value is specified, the default length of 4 will be assumed.

The LEN parameter for type H may be specified as a value of 1 or 2. If the parameter is omitted or an invalid value is specified, the default length of 2 will be assumed.

The LEN parameter for type D may be specified as a value from 1 through 8. If the parameter is omitted or an invalid value is specified, the default length of 8 will be assumed.

The LEN parameter may be specified for type C. However, if it is omitted, the number of characters, not including the enclosing apostrophes, specified on the INIT parameter will be used as the length.

The LEN parameter may be specified for type X. However, if it is omitted, the number of characters specified on the INIT parameter will be used to determine the length. If the number of characters in the INIT parameter is odd, 1 will be added to the number, and the sum is divided by 2. If the number of characters in the INIT parameter is even, the number will be divided by 2. The result of the division will be used as the length.

INIT=
Specifies the initial data to be placed on the data base for this item. If the INIT parameter is omitted and the TYPE is C, the initial

data will default to blanks. If the INIT parameter is specified and the TYPE is C, the initial data must be enclosed in single apostrophes and must conform to assembler language specifications for a character-type constant. The default data for all other types will be zeroes.

RPT=
Is a repetition factor to determine the number of times the initial data for this item will appear on the data base as part of this item. If the RPT parameter is omitted or specified as 0, the default value of 1 will be assumed. The value specified for this parameter includes the original copy of the item data. For example, "RPT=1" gives only the original item data; "RPT=2" gives the original item data plus one copy of the item data.

DISP=
Provides the capability to specify the displacement into an unblocked array or the displacement into a block of a blocked array where the initial data on the ITEM macro will start.

DEFMSG

The DEFMSG macro is used to define messages to be used by the message writer function. The maximum length of a message including variates. However, when defining a message, the length should conform to the length restrictions of the device(s) to which it will be routed.

[symbol]	DEFMSG	number, ROUTE=code $\left[,\text{ACT=}\begin{Bmatrix}\underline{\text{I}}\\ \text{A}\\ \text{D}\end{Bmatrix}\right]$ $\left[,\text{DATE=}\begin{Bmatrix}\underline{\text{NO}}\\ \text{YES}\end{Bmatrix}\right]$,TEXT='data'

number
Is a three digit (001-999) unique message number with the first digit indicating the subsystem. (The numbers from 001-099 and 800-899 are reserved for use by the Special Real Time Operating System.)

ROUTE=
Defines routing code (0-255). Codes are assigned Codes are assigned (during SYSGEN) indicating the types of output devices. Examples would be a display unit or display group, a printer or printer group. By convention the codes 1-9 are reserved for use by the Special Real Time Operating System.

ACT=
Defines action code. Codes are assigned to indicate the type of action required in connection with a message.

I -- Information (default if operand omitted).
A -- Action required.
D -- Operator/user decision required.

DATE=
Indicates whether the date will be affixed to the message during online operations.

YES -- Affix date.
NO -- Do not affix date.

TEXT=
Defines the text of the message and the variables to be supplied by the user when the message is requested during online execution. The text is a character string, enclosed in apostrophies, with the variables positioned in the string at the position they should appear in the output message. Variables are specified by coding information in the following format:

```
#cfs#
```

Where

is a delimiter character and must appear before and after the other specifications. No blanks are allowed between them.

c defines the number of characters to be occupied by this variable in the output message.

f defines the type of data conversion to be performed on the data being output.

s specifies the position of this variable in the variable list that is passed by the calling program when the message is selected for output.

The maximum number of variables allowed is 10. The maximum message length, including variables, is 255 characters. The message will be truncated by the message writer if necessary to conform to the line length restrictions of the device to which the message is routed.

The variable data is converted to alphanumeric characters as defined by the variable specification. The valid character specifications and associated conversion actions are as follows:

F The four bytes at the address specified will be converted to decimal and inserted into the message.

H The two bytes at the address specified will be converted to decimal and inserted into the message.

C Data beginning at the address specified, for the length specified in the format specification (c above) will be moved into the message. It is assumed that the data consists of 'printable' characters.

X The data beginning at the address specified is converted to hexadecimal characters and moved into the message. If the number of characters allocated in the message is even, data is converted beginning with the first 4 bits at the address specified, and each 4 bits following are converted to a character until the message field is filled. If the number of characters allocated in the message is odd, the first 4 bits of the byte at the address specified are skipped, and conversion proceeds as above.

B The data beginning with the byte specified by the address is converted, each bit being converted to a character (1 or 0). If the number of characters allocated in the message (c) is an even multiple of 8, data conversion begins with the first bit of the first byte at the address.

  If c is not a multiple of 8, c is divided by 8, and the value 1 is added to the quotient (the remainder is dropped). This determines the number of bytes (n) from which data will be converted. The right most c bits of the n byte field will be converted to characters and moved to the message.

The following figure shows how the data would appear in the message if converted according to various variable specifications. In all cases, assume that the user has passed the address of a 4-byte area which contains X'D3042F00'.

Variable Specification	Output Characters	Position of Bit Selected for Conversion
#1Xn#	3	Low-order of 4 bits of first byte
#2Xn#	D3	Entire first byte
#3Xn#	304	Low-order 4 bits of first byte and entire second byte
#6Xn#	D3042F	Entire first 3 bytes
#8Bn#	11010011	Entire first byte
#5Bn#	10011	Low-order 5 bits of first byte
#3Bn#	011	Low-order 3 bits of first byte
#16Bn#	1101001100000100	Entire first two bytes
#13Bn#	1001100000100	Low-order 5 bits of first byte and second byte

Figure 3-10. Hexadecimal and Binary Variable Descriptions

DATA BASE BDAM DATA SET COMPRESS

The data base BDAM data set compress program DPPXDBCP provides the capability to recover lost space on data base BDAM data sets.

A single execution of the compress program can be used to compress all BDAM data sets in a single data base. However, a single execution of the compress program cannot be used to compress BDAM data sets from more than one data base.

JCL requirements are:

// EXEC	PGM=DPPXDBCP		Defines program name to be executed.
//STEPLIB	DD		Defines the data set which contains the compress program.
//SYSPRINT	DD	SYSOUT=A	Output message data set.
//SYSUT1	DD		Defines a BDAM data set to be used by the compress program.

The space allocated to this data set and the data set block size must be at least as large as the largest used by a BDAM data set to be compressed.

//DBINIT	DD	Defines a data base partitioned data set.
//ANYDADD	DD	Defines a data base BDAM data set which is part of the same data base as the PDS defined by the DBINIT DD card. The DD name used here must be the same as the DD name used during execution of the offline utility. There may be a DD statement for every BDAM data set associated with the PDC defined by the DBINIT DD statement.

Example 1 is a sample set of JCL to create a data base PDS and three associated BDAM data sets. Example 2 shows a sample of how the data base data sets are referenced during execution of the offline utility program.

Example 3 is a sample of the JCL required to collapse the BDAM data sets described in Examples 1 and 2. Notice that the DD names used to describe the data base data sets in Example 3 are identical to those DD names used in Example 2.

The SYSUT1 DD statement in Example 3 allocates two cylinders of space. This corresponds to the DBINIT2 DD statement in Example 1 which is the largest BDAM data set in this data base. The SYSUT1 DD statement in Example 3 specifies a data set BLKSIZE of 13030. This corresponds to the USERDADD DD statement in Example 1 which has the largest data set BLKSIZE in this data base.

The SYSUT1 DD statement in Example 3 may be given a disposition of NEW or OLD, but must never be given a disposition of MOD.

```
// EXEC         PGM=IEFBR14
//DBINIT         DD          DSN=DATABAS1,UNIT=SYSDA,
//              DISP=(,CATLG),VOL=SER=PPL001,
//              SPACE=(CYL,(2,,10))
//              DCB=(RECFM=U,BLKSIZE=13030)
//DBINIT2        DD          DSN=DATABAS2,UNIT=SYSDA,DISP=(,CATLG),
//              VOL=SER=PPL001,SPACE=(CYL,(2)),
//              DCB=(RECFM=U,BLKSIZE=2048,DSORG=DA)
//USERDADD       DD          DSN=USERDADD,UNIT=SYSDA,DISP=(,CATLG),
//              VOL=SER=PPL001,SPACE=(TRK,(5)),
//              DCB=(RECFM=U,BLKSIZE=13030,DSORG=DA)
//ANYDADD        DD          DSN=ANYDADD,UNIT=SYSDA,DISP=(,CATLG),
//              VOL=SER=PPL001,SPACE=(TRK,(5)),
//              DCB=(RECFM=U,BLKSIZE=2048,DSORG=DA)
```

EXAMPLE 1: Creating Data Base Data Sets

Note: The underlined portions of the JCL in Examples 1, 2, and 3 are the only portions which may be changed from the way they appear.

```
// EXEC  PGM=DPPXUTIL
     .
     .
     .
//DBINIT        DD  DSN=DATABAS1,DISP=OLD
//DBINIT2       DD  DSN=DATABAS2,DISP=(MOD,PASS),DCB=DSORG=DA
//USERDADD      DD  DSN=USERDADD,DISP=(MOD,PASS),DCB=DSORG=DA
//ANYDADD       DD  DSN=ANYDADD,DISP=(MOD,PASS),DCB=DSORG=DA
     .
     .
     .
//SYSIN         DD *
```

EXAMPLE 2: Offline Utility use of Data Sets

```
// EXEC         PGM=DPPXCBCP
//STEPLIB   DD  DSN=VAN370.LMLIB,DISP=SHR
//SYSPRINT  DD  SYSOUT=A
//SYSUT1    DD  DSN=&&SYSUT1,UNIT=SYSDA,SPACE=(CYL,(2)),
//          DCB=(RECFM=U,FLKSIZE=13030,DSORG=DA)
//DBINIT    DD  DSN=DATABAS1,DISP=OLD
//DBINIT2   DD  DSN=DATABAS2,DISP=OLD,DCB=DSORG=DA
//ANYDADD   DD  DSN=ANYDADD,DISP=OLD,DCB=DSORG=DA
//USERDADD  DD  DSN=USERDADD,DISP=OLD,DCB=DSORG=DA
```

EXAMPLE 3: Compress Program JCL

CHAPTER 4. OPERATOR'S REFERENCE

INTRODUCTION

This section of the manual is intended to be used at the CPU main console as an operator's manual. Certain information that is normally found in this section of a Description and Operations Manual is documented in the previous chapter in the section entitled, "Pre-SYSGEN Initialization" and will not be duplicated.

The Operator's Reference contains the Special Real Time Operating System operation information. It is to be used as part of the operator's library. This section is intended for the system operator, but some sections are also of interest to operators at secondary consoles or terminals.

The user of this section should have a thorough knowledge of OS/VS1 operation. This section is not intended to replace OS/VS1 operator reference material.

The JCL required for realtime execution will vary greatly for each account and will most likely be provided to the operator by the system programmer. Therefore, the JCL requirements are documented in the section entitled, "System Initialization".

The information in this section is intended to assist the operator running the Special Real Time Operating System and in diagnosing the Special Real Time Operating System control statement errors. It is organized as follows:

- Normal Operating Procedures
- Control Card Information
- Two-Partition Operation
- Failover/Restart
- Normal Termination Procedures
- Abend Codes
- The Special Real Time Operating System Messages

NORMAL OPERATING PROCEDURES

The Special Real Time Operating System executes as an OS/VS1 job which does not terminate normally. It is entered into the system through JCL just as any OS/VS1 job. The Special Real Time Operating System has its own messages with operator action for some, these are described in this section under messages.

After the Special Real Time Operating System job has been started and the OS job started, message appears, the Special Real Time Operating System issues a WTOR 'SRTOS INPUT MESSAGE PROCESSING AWAITING REPLY', and leaves the reply outstanding. This allows the operator to communicate with the Special Real Time Operating System. The commands which are defined as part of the Special Real Time Operating System are:

```
CANCEL,operands
REPORT,operands
DREC,operands
DDSCNTRL,operands
DLMP,operands
MSGRC,operands
QS,operands
STAE,operands
STOP
```

Each of these commands requests a specific service. Other commands may be defined by other PRPQs or program products or by the user through the IMP macro of SYSGEN.

The operator may enter a command by replying to the WTOR in the following format:

$$\text{r xx, 'command}\left[\left\{\begin{array}{l}\text{, SLAVE [, P1, P2, . . . , Pn]}\\ \text{[, P1, P2 . . . Pn]}\end{array}\right\}\right]\text{'}$$

R xx is the format required by OS/VS1, and xx is the message number to which the reply responds. The content of the reply will be in a standard format. The command verb is the first element of the entry. It may be selected from any valid IMP command defined at SYSGEN.

If two-partition operation is SYSGENed and active, the keyword SLAVE may be included as the second element. If included, the command will be processed in the SLAVE partition. If SLAVE is omitted or two partition operation is not SYSGENed, the command will be processed in the master partition.

Parameters to be associated with the command are entered following the command verb or the keyword SLAVE, separated by commas.

The keyword SLAVE is not referenced in the following command descriptions. Unless specifically disallowed, the command may be entered to be processed in the slave partition by including the keyword following the command verb.

CANCEL Command

The CANCEL command should be used to terminate a Special Real Time Operating Sys tem job.

The format of the command and its operands is:

$$\text{r xx,'CANCEL}\left[\begin{Bmatrix}\text{,DUMP}\\ \text{,NODUMP}\end{Bmatrix}\right][\text{,COMMENTS}]\text{'}$$

CANCEL
Informs the input message processing routine that this reply is to cancel the Special Real Time Operating System jobstep for which this reply is outstanding.

DUMP
Cancel Special Real Time Operating System with a dump. The ABEND (dump) code is a user 122.

NODUMP
Cancel the Special Real Time Operating System without a dump. The ABEND code is a user 222.

Comments
Operator explanation as to reason for cancelling the Special Real Time Operating System. MESSAGE 60 will be issued, before cancelling the Special Real Time Operating System, containing the comment. The maximum length of the comment is 80 characters.

REPORT Command

The REPORT command is used to control the execution of the Report Data Output facility of the Special Real Time Operating System. The format of the REPORT command is:

$$\text{r xx, REPORT} \left[, \begin{Bmatrix} \text{ADD} \\ \text{NEW} \end{Bmatrix} \right] \text{, output ddname, input ddname}$$
$$\text{[, input ddname, input ddname, . . .]}$$

REPORT
Informs the input message processing routine that this reply is for the Report Data Output Facility.

NEW
Report Data Output Facility will begin writing data at the beginning of the output data set.

ADD
Report Data Output Facility will begin writing data at the end of the output data set.

Output ddname
A ddname which points to a QSAM data set to be used as an output data set. The record length of the data set must be equal to or greater than the maximum record length of the input data sets.

Input ddname
A ddname which points to a QSAM data set to be used as an input data set. A maximum of 10 input DDNAMES may be specified.

DREC Command

The DREC command is used to control the execution of the Special Real Time Operating System Data Record and Playback function. The format of the DREC Command is:

$$\text{r xx, DREC}\left\{\begin{array}{l}\text{, ENABLE}\left\{\begin{array}{l}\text{, ADD}\\ \text{, DEL}\\ \text{, ALL}\end{array}\right\}\text{, ID, ID, ID, . . .}\\ \\ \text{, DISABLE}\end{array}\right\}$$

DREC
Inform the input message processing routine that this reply is to initialize data recording.

ENABLE Initialize data recording.

DISABLE
Deinitialize data recording.

ADD
Add the following ID's to the data recording table.

DEL
Delete the following ID's from the data recording table.

ALL
Enable all data recording IDs.

ID
Three digit hex numbers (001-FFF) that must be used in recording data as (enable ID's).

DDSCNTRL Command

The DDSCNTRL Command is used to control the functions of duplicate data set support. The format of the DDSCNTRL command is:

```
                            ┌ ( ,TAKE                                          ) ┐
                            │ | ,REPLACE [PRIMARY WITH YYYYYYYY]               | │
r xx,DDSCNTRL {,XXXXXXXX}   │ < ,SWITCH                                        > │
                            │ | ,CREATE                                        | │
                            │ | ,STATUS   [,[ddname1][,ddname2]]               | │
                            └ ( ,COMPARE                                       ) ┘
```

The DDSNAME specified (or defaulted) on the DDSNAMES input control card which declared this duplicate data set.

TAKE
Causes the backup to be taken out-of-service.

REPLACE PRIMARY WITH YYYYYYYY
Causes the primary to become the backup (still in service), and sets the data set with DDNAME YYYYYYYY to become primary. (YYYYYYYY will default to the old backup.) This function requires that the DDS DCBs be closed.

CREATE
Causes the primary to be copied track-by-track onto the backup, and sets the backup to be in-service (requires backup to be out-of-service on request).

SWITCH
Causes the backup to become the primary and sets the primary as the backup out-of-service. (Requires backup to be in-service on request).

STATUS
Prints a message (#56) stating primary and backup DDNAMES, and if backup is out-of-service.

COMPARE
Invoke the IEBCOMPR utility against ddname1 and ddname2. ddname1 will default to the primary DDNAME and ddname2 will default to the backup DDNAME.

DLMP Command

The DIMP Command controls the operation of the Dynamic Load Module Purge feature of the Special Real Time Operating System. It permits the system operator to cause a load module to be deleted from virtual storage which has been loaded by the Special Real Time Operating System task management in response to PATCH requests.

The format of the command and its operands is:

```
r xx,DLMP,time,module-name,module-name...
```

DLMP
This input command is for the dynamic load module purge feature.

time
Decimal integer value between 0 and 1200; time in seconds that the DLMP feature will wait to allow other tasks to complete execution of the specified load modules. If time is omitted, a default value of 2 seconds will be used.

module name
Name of the load module(s) that is to be purged from virtual storage. Up to 10 module names, separated by commas, may be specified in one request.

MSGRC COMMAND

The MSGRC command is used to place a system message routing code in or out of service, determine the status of one or more routing codes, or change the alternate routing code.

```
                   (    IN    )
                 ( |   OUT    |
r  xx,MSGRC, {rc}{ ,  STATUS  } [,altrc]
             { 0}{ , STATALL  }
                   (          )
```

MSGRC
Informs the input message processing routine that this reply is for the Message Routing Code Status Change Facility.

Note: This command should not be entered in the SLAVE partition.

rc
Routing code.

0
This parameter is 0 if STATALL is specified.

IN
Place RC in service.

OUT
Place RC out of service.

STATUS
Display the status, via a system message, of the specified routing code (RC).

STATALL
Display the status, via a system message, of all the routing codes in the system.

altrc
This parameter is recognized only if IN or OUT is specified.

altrc is the routing code to which messages are directed should the primary routing code be out of service or the output operation fail.

QS COMMAND

The QS command is an IMP command which may be used to display or alter the status of queue holders and queue processors. The format of the command and its operands is:

```
             /QPnn \     / SEQ      \
             |ALLQP|     | NONSEQ   |
             {name }     { HOLD     }
   r xx,QS,  |ALLQH|  ,  | REL      |     [,PURGE]
             \ALL  /     | NOPATCH  |
                         | STATUS   |
                         \ XREF     /
```

Where the first positional parameter, QS, is the input message processing command name for the queue status facility, the second positional parameter is a noun used to identify the queue holders, queue processors, or independent task, the third positional parameter is a verb used to specify the primary action to be taken, and the fourth positional parameter is a verb used to specify the secondary action to be taken. The first three positional parameters are required. The fourth positional parameter is optional.

QPnn
The numberic value, nn, of this noun identifies a specific queue processor to be serviced on this command. The queue processor is identified by the numberic value specified on the QP statement in the SYSINIT input stream. This ID has a range 00 to 99. Two characters must be supplied.

ALLQP
This noun is used to indicate that all queue processors are to be serviced on this command.

name

The 1 to 8 character name of this noun idnetifies a specific queue holder or independent task to be serviced on this command. The queue holder is identifed by the name specified on the QH statement in the SYSINIT input stream or an independent task by the name specified by the PATCH which created it.

ALLQH
This noun is used to indicate that all queue holders are to be serviced on this command.

ALL
This noun is used to indicate that all queue holders, queue processors, and independent tasks are to be serviced on this command.

SEQ
This verb is used to request that work queued to the specified queue holder(s) be processed sequentially, (i.e., whenever work has been selected by a queue processor from a sequential queue holder, no other queue processor may select work from that queue holder until that work has been completed). Entry of this command will have no effect on any work that has already been selected by any queue processors.

NONSEQ
This verb is used to request that work queued to the specified queue holder(s) be processed non-sequentially (i.e., two or more queue processors may process work from that queue holder concurrently.). A NONSEQ QS command will have no effect on any work that has already been selected by a queue processor.

HOLD

This verb is used to request that the specified queue holders, queue processors and/or independent tasks be held. As a result, no work may be selected from the specified queue holder(s) by any queue processor, the specified queue processor(s) will be prohibited from selecting work from any queue holder and specified independent task(s) will be prohibited from starting processing for any work that may be queued. Work may be added to the specified queue holder(s) or independent task(s) until the maximum queue length has been reached. A HOLD QS command will have no effect on any work that has already been selected for processing.

REL

This verb is used to release the work from the HOLD state the specified queue holders, queue processor and/or independent tasks for normal processing. A REL QS command will have no effect on any work that has already been selected by a queue processor.

NOPATCH

This verb is used to request that the specified queue holder(s) or independent task(s) to not accept additional PATCHes. Any PATCHes executed to a queue holder or independent task in a NOPATCH state will be rejected and condition code 6 will be returned to the user. Any work previously queued to the specified queue holder(s) and/or independent task(s) will be processed normally and will not be effected by a NOPATCH QS command.

PATCH

This verb is used to request that the specifeid queue holder(s) and/or independent task(s) may now accept PATCHes.

STATUS

This verb is used to request that the current status of the specified queue holder(s), queue processor(s) and/or independent task(s) be displayed. This information is output as message 862 and includes: TCBX name, element type (QP, QH or TSK), PATCH/NOPATCH status, HOLD/REL status, SEQ/NONSEQ status, current queue length and the character 'A' if the task for a QP or independent task is currently processing a work queue.

XREF

This verb is used to request that the current status of the specified queue holder(s), queue processor(s) and/or independent tasks to be displayed. This information includes message 862 as defined for the STATUS verb and in addition, message 863 will be output one or more times for each queue holder and/or queue processor specified. It includes, for a queue holder, the names of the queue processors which may select work from it and for a queue processor, the names of the queue holders from which it may select work.

PURGE

This verb may be used in conjunction with any primary verb to request that any work previously queued to the specified queue holder(s) and/or independent task(s) be purged by a PURGEWQ macro. A PURGE command will have no effect on any work that has already been selected by a queue processor.

STAE COMMAND

The STAE command may be used to suppress system ABEND dumps for all or selected load modules.

The format of the command and its operands is:

```
              ┌ ┌         ┐ ┐
              │ │,DUMP    │ │
 r xx,STAE    │ │,NODUMP  │ │  ,modename1,...,modname n
              │<│,ONEDUMP │>│
              │ │,STEP    │ │
              │ │,OPTION  │ │
              └ └         ┘ ┘
```

STAE
Is a required positional operand which informs the input message processor that his reply is for the Dump/No Dump facility.

DUMP
Allow a dump to be taken for these modules (provided there is a SYSUDUMP or SYSABEND DD statement).

NODUMP
Suppress a dump from being taken for these modules.

ONEDUMP
Allow one dump to be taken for these modules (provided there is a SYSUDUMP or SYSABEND DD statement) but suppress any additional dumps for that module.

STEP
ABEND the job step if one of these modules ABEND.

OPTION
Allows the operator to choose whether or not to take a dump following an ABEND of these modules. The operator is informed of the ABEND via a WTQR (message 850) and must reply 'YES' to receive the dump. If no reply is issued in five minutes, the dump is automatically suppressed.

modname1,...,modname n
Is used to indicate the load module(s) that are to be covered by the specified option. A maximum of 10 load module names may be specified on any one reply. Null fields (double commas) will not be accepted.

If no load module names are specified, the mode defined by the previous parameter will apply to all load modules.

STOP Command

```
 r xx,STOP
```

STOP
Cancel the Special Real Time Operating System without a dump. The ABEND code is a user 222.

CONTROL CARD INFORMATION

The format of the Special Real Time Operating System control statements is very similar to the format of JCL statements. That is, there are four standard fields in the card. They are:

LABEL OPERATION OPERANDS COMMENTS

where: LABEL
Is the control statement label and must begin in column one. If column one contains an asterisk (*), the entire card is a comment card. The LABEL cannot exceed eight characters and must be separated from the OPERATION by at least one blank. The LABEL field is optional.

where: OPERATION
Is the type of action that the card represents. This field must contain one of the following:

QP
QH
PATCH
WAIT
RESTART
TCB
GETWA
CBGET
ABEND
MASTER
SLAVE
DBREF
STAEX

The OPERATION field must be separated from the LABEL (if any) and OPERANDS by at least one blank. If no LABEL exists the OPERATION must not start in column one. The OPERATION field is required.

where: OPERANDS
The OPERANDs field is required for all OPERATION types except ABEND and must begin on the same card as the OPERATION. Each OPERATION has unique OPERANDs (see System Initialization). The OPERANDs must be separated from the OPERATION and COMMENTs by at least one blank. There is a limit of 255 OPERAND characters for one OPERATION.

where: COMMENTS
The COMMENTs are optional and there is no limit to the length of COMMENTs allowed. The COMMENTs must be separated from the OPERANDs by at least one blank.

CONTINUATION

Control cards may be continued. Continuation is requested by Continuation is requested by ending the OPERAND's field with a comma or putting a nonblank in column 72, or both. If the OPERANDS are completed and COMMENTs are to be continued, column 72 must be nonblank.

An example of valid continuations follows:

```
                                                                 COL 72
P1      PATCH    EP=TEST,                                              *
                 TASK=TEST

P2      PATCH    EP=TEST,
                 TASK=TEST

P3      PATCH    EP=TEST                                   TEST PROGAM*
                 AND RETURN
```

Continuation cards must begin in column 16. PATCH cards with PARAM data containing blanks within quotes may be continued to the next card. The continuation is assumed to start in column 16.

EXAMPLE:

```
                                                                 COL 72
  PTCH PATCH      EP=TEST,PARAM=(C'ABCbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb*
       bbbbbbbbbXYZ')


       COL16
       The PARAM data would be
           'ABCbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbXYZ'
```

TWO-PARTITION OPERATION

The presence of a MASTER or a SLAVE statement in the input stream signifies two-partition operation. When this occurs, the first job started will issue the message DPPO46I and will repeat the message at one-minute intervals until the other job has started. The operator should ensure that the job names of the SLAVE corresponds to the na me given on the MASTER card by the SLAVE=jobname operand and that the jobname of the MASTER corresponds to the name given on the SLAVE card by the MASTER=jobname operand. If the MASTER job terminates, the SLAVE will also be terminated with a USER 41 ABEND code. However, if the SLAVE job terminates and the MASTER is still executing, the SLAVE job may be restarted in the same or another partition.

Note: The Special Real Time Operating System job should not be terminated by a CANCEL JOBNAME command as STAE processing will be bypassed. As a result, the SLAVE will not be terminated when the MASTER ends, and the SLAVE will not restart if an attempt is made to restart it.

FAILOVER/RESTART OPERATION

SINGLE CPU ENVIRONMENT

The operator may cause a RESTART by dialing the device address which contains the RESTART data set into the 'LOAD UNIT ADDRESS' switches of the CPU and depressing LOAD (IPLing). This operation will be successful only if a RESTART data set had been previously written by a RESTART WRITE statement. The data set would have been written to the data set allocated to the DPPFAIL DD card. If copies had been made of the DPPFAIL data set, a RESTART could be caused by the operator IPLing the DA device containing the DPPFAIL data set or a copy of the DPPFAIL data set.

SINGLE CPU ENVIRONMENT WITH CONTINUOUS MONITOR

The continuous monitor is a program which monitors the online CPU and, if any failures are detected, recommends a RESTART. The RESTART is recommended by message DPP098. When this message is issued, the operator must RESTART the system as described above.

TWO-CPU ENVIRONMENT WITH CONTINUOUS MONITOR AND PROBE

With the continuous monitor operating in the realtime CPU and the PROBE in the backup CPU, no operator intervention is required on a FAILOVER condition. The FAILOVER is initiated by the PROBE when an error condition is detected. If, however, the system has a Computer Status Panel installed, and the status panel is not switched to auto mode, the operator must decide whether to cause a FAILOVER to the backup CPU or to RESTART in the online CPU. He must then initiate the action by depressing the SELECT button for the CPU which he wants to execute as the online CPU.

NORMAL TERMINATION PROCEDURES

The Special Real Time Operating System should be terminated by replying to the Input Message, Processor's outstanding message with a CANCEL or STOP command. This command and its operands are documented earlier in this chapter. The Special Real Time Operating System should not be terminated by the OS/VS1 CANCEL command which causes the STAE processing for the job step task to be bypassed. The effect of bypassing the STAE may be to leave certain system functions in a condition which will degrade subsequent system operation. If it is necessary to use the OS/VS1 CANCEL command, the OS/VS1 system should be re-IPLed at the earliest convenience.

The Special Real Time Operating System should be terminated with the OS CANCEL command only as a last resort. Terminating the Special Real Time Operating System with the OS CANCEL command bypasses STAE cleanup routines and will not cause the SLAVE job to be terminated when the MASTER ends and the MASTER may cause processing errors for any job that starts in the partition the SLAVE ended in. If a SLAVE job is terminated with the OS CANCEL command, it will not be able to restart.

If it`is necessary to use the OS CANCEL to terminate a MASTER job, the SLAVE must also be terminated with the OS CANCEL command.

THE SPECIAL REAL TIME OPERATING SYSTEM ABEND CODES

USER 001 Issued by: DPPITIMI

Explanation: An invalid TCBX address was found in the job step TCBUSER field.

Action: Restart the system.

USER 002 Issued by: DPPITIMI

Explanation: An invalid SCVT or XCVT address was found in the job step control block chain.

Action: Restart the system.

USER 003 Issued by: DPITIMI1

Explanation: The time-of-day (TOD) clock was not operational.

Action: Probable hardware failure.

USER 004 Issued by: DPPITIMI

Explanation: Time array - DPPCTIMA - was not found in the data base.

Action: DD card DBINIT must allocate a data base data set that contains a DPPCTIMA array.

USER 010 Issued by: DPPIDBAS

Explanation: An invalid TCBX address was found in the job step TCBUSER field.

Action: Restart the system.

USER 011 Issued by: DPPIDBAS

Explanation: An invalid SCVT or XCVT address was found in the job step control block chain.

Action: Restart the system.

USER 012 Issued by: DPIDBAS1,DPIDBAS2, or DPIDBAS3

Explanation: Data base initialization was unable to open one of the following data sets:
DDname - DBINIT
DDname - DBINIT2
or a user specified data base data set.

Action: DD cards must exist for DBINIT and DBINIT2 and user-requested data base sets Be sure that the data set exists.

USER 013 Issued by: DPIDBAS1

Explanation: Data base initialization was unable to find member @INIT in the DBINIT data set via BLDL.

Action: DD card DBINIT must allocate the correct data base data set containing member name @INIT.

USER 020 Issued by: DPPMINIT

Explanation: The online message handler initialization was unable to OPEN the message data set DCB.

Action: The job's JCL must contain a DD statement with DD name MSGDS.

USER 022 Issued by: DPPINIT1

Explanation: This ABEND was requested by the user through the presence of an ABEND statement in the SYSINIT input stream.

Action: None.

USER 023 Issued by: DPPMINIT

Explanation: The online message handler initialization could not find the message routing code array DOMXSMRC in the data base.

Action: The array should be generated at the Special Real Time Operating System SYSGEN time by the use of the MSGRC macro. DMINIT must have a DD card that allocates the correct data set.

USER 030 Issued by: DPPINIT

Explanation: The Special Real Time Operating System initialization (DPPINIT) was not running under the job step task TCB.

Action: Program DPPINIT cannot be attached, but must be the job step task.

USER 031 Issued by: DPPINIT1

Explanation: A PATCH macro was executed in response to a PATCH initialization card. The return code from the PATCH macro was greater than 4 (i.e., the PATCH failed).

Action: At the time of the dump, Register 3 contains the address of a PATCH control block. Four bytes past that address is the address of the PATCH PROBL and X'14' bytes past that address is the PATCH SUPL. The first 9 bytes of the SUPL contain the task name and the second 8 bytes contain the entry point name specified on the PATCH card for which the failure occurred. Register 15 contains the PATCH return code. Make appropriate corrections and retry.

USER 032 Issued by: DPPFIXFR

Explanation: An invalid address range was passed to be either fixed in real storage or unfixed.

Action: Correct the address range and retry, ensure that array DPPXFIX is valid.

USER 033 Issued by: DPINIT3

Explanation: Initialization was unable to get enough control block (CBGET) storage in which to create a TCBX at initialization time.

Action: Increase the CBGET storage with a CBGET statement in the SYSINIT input stream and retry.

USER 034 Issued by: DPINIT05

Explanation: A syntactical error was detected on one or more of the initialization input (SYSINIT) statements.

Action: Correct the statement(s) in error and retry.

USER 035 issued by: DPPINIT1

Explanation: A pre-WRITE RESTART program which was PATCHed as a result of a PATCH statement in the SYSINIT input stream completed and returned with a non-zero POST code.

Action: Correct the failing program and retry.

USER 036 Issued by: DPINIT5

Explanation: On a two-partition run, a MASTER or SLAVE partition had been posted by the other partition; however, the corresponding jobname could not be found.

Action: Correct the MASTER or SLAVE card so that the operands give the exact jobname of the job in the corresponding partition.

USER 037 Issued by: DOMIRFLV

Explanation: During an attempt to write a FAILOVER data set (WTFAILDS), the SLAVE partition could not be found.

Action: If the SLAVE job has ABENDED or terminated, fix the failure and resubmit the run.

USER 038 Issued by: DOMIRFLV

Explanation: Multiple simultaneous attempts were made to write a FAILOVER data set (WTFAILDS) from the same job.

Action: Correct the programs.

USER 039 Issued by: DOMIRFLV

Explanation: The WTFAILDS macro was issued by a non-real time job.

Action: The WTFAILDS macro must be issued by a Special Real Time Operating System job.

USER 040 Issued by: DPINIT05

Explanation: The SYSINIT DCB could not be opened, no SYSINIT DD card was provided or a SYSINIT stream was processed that did not contain a PATCH statement.

Action: The job's JCL must contain a SYSINIT DD card and at least one PATCH statement in the input stream.

USER 041 Issued by: DPPISTAE

Explanation: The SLAVE job is abnormally terminated with this code when the MASTER job step terminates. The SLAVE cannot continue to run because it does not have full Special Real Time Operating System services.

Action: Restart MASTER and SLAVE jobs.

USER 042 Issued by: DPINIT5

Explanation: During the restart of a SLAVE job, the initialization routine could not locate the job named on the MASTER= operand of the SLAVE statement.

Action: Correct the SLAVE statement and retry.

USER 043 Issued by: DPINIT5

Explanation: The SLAVE job was being restarted after a failure; during initialization the SLAVE initialization found the MASTER job step to be terminating.

Action: Restart both MASTER and SLAVE jobs.

USER 044 Issued by: DPINIT5

Explanation: An attempt was made to restart a SLAVE for a MASTER which already had a SLAVE job executing.

Action: Verify the jobname on the SLAVE MASTER=jobname control statement. It should have a jobname for a MASTER job step that does not have a SLAVE job currently executing.

USER 045 Issued by: DPPINIT1

Explanation: A RESTART statement was processed in the input stream which requested the CANCEL function.

Action: None.

USER 046 Issued by: DPPINIT1

Explanation: The maximum size GETWA space allocated was not sufficient to satisfy the requirements of the Special Real Time Operating System. A GETWA size of 1024 bytes is required.

Action: Allocate a larger GETWA size (1024 bytes) on the GETWAS parameter of the VS macro and regenerate the system or specify appropriate GETWA sizes on the GETWA statement in the SYSINIT input stream and rerun the job.

USER 050 Issued by: DPXDBIN6

Explanation: The offline utility program was unable to OPEN the initialization data set for DD name DBINIT.

Action: The offline utility JCL must include a DBINIT DD card.

USER 051 Issued by: DPXDBIN4

Explanation: The offline utility program had an error while attempting to STOW initialization member @INIT.

Action: Retry.

USER 052 Issued by: DPXDBIN1

Explanation: The offline utility program was unable to obtain initialization member @INIT from the initialization data set.

Action: Retry run.

USER 053 Issued by: DPXDBIN2

Explanation: A loggable array was created which named a log array. The named log array could not be found.

Action: Recompile the loggable array to have the log array recreated.

USER 054 Issued by: DPIDBAS3

Explanation: A BLDL error was encountered while attempting to read the PDS directory entry for a loggable array.

Action: Recompile to loggable array to have the log array recreated.

USER 055 Issued by: DPPXUTIL

Explanation: The assembler encountered an error and returned an error code greater than 16.

Action: Correct the problem indicated by the assembler output and retry the job.

USER 064 Issued by: DPPTETXR

Explanation: Program DPPTETXR has been entered under a TCB which is not a job step TCB.

Action: If a program is linking to DPPTETXR, correct and retry, otherwise retry.

USER 065 Issued by: DPPTDSVC

Explanation: A DPATCH=I was issued for the specified task so the Special Real Time Operating System task management terminated it with this code.

Action: None.

USER 071 Issued by: DPPXDPB

Explanation: Data playback was unable to OPEN the data playback DCB.

Action: A DD card named DPBIN must exist if data playback is to be used.

USER 072 Issued by: DPPXDPB

Explanation: The BLKSIZE and LRECL on the DPBIN DD card is smaller than the maximum BLKSIZE and LRECL used when the data recording/playback data set was defined and/or when data was recorded on the data recording/playback data set.

Action: The BLKSIZE and LRECL on the DPBIN DD card must be equal to the maximum BLKSIZE and LRECL on the data recording/playback data set.

USER 080 Issued by: DPPSINIT

Explanation: A bad card was found in the DDSCTLIN input stream.

Action: Correct control card and retry.

USER 081 Issued by: DPPSCHCK

Explanation: User received software I/O error but did not specify a SYNAD exit routine for a DDS data set.

Action: None.

USER 122 Issued by: DPPXKILL, DPPXIMPW

Explanation: The job step task has been ABENDed due to the CANCEL, DUMP request.

Action: None.

USER 222 Issued by: DPPXKILL

Explanation: The job step task has been ABENDed due to a CANCEL, NODUMP request.

Action: None.

SYSTEM 4xx Issued by: A USER-GENERATED SVC

Explanation: A user program made an invalid SVC The request was for the SVC number xx. xx is the hexadecimal number of the SVC which was issued.

Action: Correct program and ensure that a valid SVC request is made.

SYSTEM 6xx Issued by: A USER-GENERATED SVC

Explanation: A user program made a Special Real Time Operating System SVC request from a non-Special Real Time Operating System job step task. The SVC requested is indicated by xx. xx is the hexadecimal number of the SVC which was issued.

Action: Check the user program and be sure that services which are not available to a non-Special Real Time Operating System task are not requested by a non-Special Real Time Operating System task.

DPP009I POST-RESTART DATA BASE AND PRE-RESTART DATA BASE ARE DIFFERENT

Routing code: 1

Message issued by segment: DPPDWRST

Explanation: One or more data base arrays have been recompiled onto the DBINIT data set that is online at restart time since the restart data set was written or a different data base is being referenced. Results are unpredictable and continued operation may or may not be successful.

Response: None.

DPP010I Time date DPPTETXR * EXCEPTL CONDITION BAD WQE XXXXXXXX

Routing code: 2

Message issued by segment: DPPTETXR load module DPPTETXR

Explanation: A subtask running the PATCH monitor DPPTPMON terminated and caused the End-of-Task-Exit Routine to be entered. The address of the current WQE XXXXXXXX was found to be invalid. Therefore, DPPTETXR cannot perform its full service which causes CB-GET storage to be lost.

Response: None.

DPP011I time DPPTETXR * ABEND IN MESSAGE OUTPUT TASK

Routing code: as defined through SYSGEN

Message issued by segment: DPPTETXM load module DPPTETXR

Explanation: The Special Real Time Operating System message output task DPPMMSG1 ABENDed and caused the End-of-Task-Exit Routine to be entered. This message is issued through a WTO macro, because the message output task is not available to print/display it.

Response: None.

DPP012I time date DPPTETXR * TASK TTTTTTTT ENDED WITH CC XXXXXXXX
or
DPP012I time date DPPTETXR * TASK TTTTTTTT ENDED WITH CC XXXXXXXX WQID NNN PATCH EP EEEEEEEE

Routing code: 2

Message issued by segment: DPPTETXR load module DPPTETXR

Explanation: The Special Real Time Operating System task TTTTTTTT terminated and caused the End-of-Task-Exit Routine to be entered. The TCB completion code field was XXXXXXXX.

If the address of the current WQE is available to the routine, it also displays the ID NNN and the EP name EEEEEEEE that was specified when the originating PATCH was issued.

Response: None.

DPP013I time date DPPTETXR * EXCEPTL CONDITION BAD TCBX XXXXXXXX

Routing code: 2

Message issued by segment: DPPTETXR load module DPPTETXR

Explanation: A subtask running the PATCH monitor DPPTPMON terminated and caused the End-of-Task-Exit Routine to be entered. The address of the TCB extension XXXXXXXX, which is contained in the TCB USER field, was found invalid. Therefore, DPPTETXR cannot perform its full service. This causes loss of CB-GET storage for TCBX, WQE chain, and LCB chain and may also result in system degradation.

Response: None.

DPP014I time date DPPTPMON * TASK TTTTTTTT EP EEEEEEEE WQID NNN NOT FOUND BY BLDL

Routing code: 2

Message issued by segment: DPTPMON3 load module DPPTPMON

Explanation: A PATCH macro was issued for task TTTTTTTT that specified an ID of NNN and an EP name of EEEEEEEE. The PATCH monitor DPPTPMON issued BLDL for the given EP name and received a return code of 4, which indicates that the module was not found. If an ECB = address was specified with PATCH, that ECB is posted with a completion code of 48 in the high order byte.

Response: None.

DPP015I time date DPPTPMON * TASK TTTTTTTT EP EEEEEEEE WQID NNN BLDL I/O ERROR

Routing code: 2

Message issued by segment: DPTPMON3 load module DPPTPMON

Explanation: A PATCH macro was issued for task TTTTTTTT that specified an ID of NNN and an EP name of EEEEEEEE. The PATCH monitor DPPTPMON issued BLDL for the given EP name and received a return code of 8, which indicates that a permanent I/O error was detected when the OS/VS1 system attempted to search the directory. If an ECB= address was specified with PATCH, that ECB is posted with a completion code of 48 in the high order byte.

Response: None.

DPP016I time date DPPTPMON * TASK TTTTTTTT NOT FOUND ON ACTIVE CHAIN

Routing code: 2

Message issued by segment: DPTPMON1 load module DPPTPMON

Explanation: The PATCH monitor attempted to remove a TCB extension from the active task chain and could not find it on that chain.

Response: None.

DPP017I time date DPPTSMON * NO LOAD REQUEST FOUND

Routing code: 2

Message issued by segment: DPTSMON1 load module DPPTSMON

Explanation: The system monitor DPPTSMON was posted and attempted to service a request for LOAD of a reentrant load module, which is indicated by a flag in the TCBX. However, when trying to find the LCB for which the LOAD was requested, no LCB with the LOAD request flag turned on could be found on the TCBX-LCB chain. DPPTSMON POSTs the PATCH monitor to continue its processing.

Response: None.

DPP018I time date DPPTETXR * TASK TTTTTTTT EP EEEEEEEE WQID NNN DID NOT RETURN TO DPPTPMON

Routing code: 2

Message issued by segment: DPPTETXR load module DPPTETXR

Explanation: A PATCH macro was issued for task TTTTTTTT that specified an ID of NNN and an EP name of EEEEEEEE.

The PATCH monitor scheduled that WQE and passed control to the specified module, which did not return control to DPPTPMON as required in the Special Real Time Operating System environment.

The module returned control to OS/VS1 which scheduled the End-of-Task-Exit routine.

If an ECB= address was specified with PATCH, the ECB is posted with 4C in the high order byte.

Response: Check the module for possible
SVC 3 EXIT
ABEND 0
LINK
XCTL
Change to BR to return control to DPPTPMON.

DPP019I time date DPPTDLMP * TIME VALUE TOO HIGH REQUEST ABANDONED

Routing code: 2

Message issued by segment: DPPTDLMP load module DPPTDLMP

Explanation: On an input command DLMP to Dynamic Load Module Purge, the time specified was not between 0 and 1200 seconds.

Response: Use a valid time value and issue the command again.

DPP020I time date DPPTDLMP * LOAD MODULE PURGE ENTERED.

Routing code: 2

Message issued by segment: DPPTDLMP load module DPPTDLMP

Explanation: A valid DLMP command has been accepted and the Dynamic Load Module Purge function is in progress.

Response: None.

DPP021I time date DPPTDLMP * MODULE MMMMMMMM DID NOT COMPLETE IN TIME FOR PURGE

Routing code: 2

Message issued by segment: DPPTDLMP load module DPPTDLMP

Explanation: A DLMP command was issued but the module MMMMMMMM did not finish executing before the time specified on the command had expired. Message DPP022I will follow to indicate the end of the purge function.

Response: One of the following:

- Retry command after module completes execution, if known.
- Try again using a higher time value.
- If module MMMMMMMM is a long running program, it may be impossible to purge it at all.
- If module MMMMMMMM is either in an endless loop or in a WAIT state, it may be impossible to purge it.

DPP022I time date DPPTDLMP * LOAD MODULE PURGE ABANDONED

Routing code: 2

Message issued by segment: DPPTDLMP load module DPPTDLMP

Explanation: A DLMP command was issued but could not be completed. This message follows other explanatory messages and indicates the end of the purge function.

Response: Check for previous messages DPP021I.

DPP023I time date DPPTDLMP * LOAD MODULE PURGE COMPLETE

Routing code: 2

Message issued by segment: DPPTDLMP load module DPPTDLMP

Explanation: A DLMP command was issued and executed successfully. This message indicates the end of the purge operation.

Response: None.

DPP024I STAE OPTION xxx IS yyy zzz

Routing code: 2

Message issued by segment: DPPTIMPS

Explanation: This message is issued to reply to the Input Message Processor of the form:

R xx,STAE,...

It is used to notify the operator of the results of the STAE command. xxx is the option selected on the STAE command. yyy is an indication as to whether the STAE command was valid (yyy = IN EFFECT) or erroneous (yyy = INVALID).

zzz further defines the result of the STAE command:

zzz = FOR ALL LM. ALL PREVIOUS STAE REQUESTS ARE CANCELLED indicates that the general STAE command was accepted.

zzz = FOR VALID LM NAMES SPECIFIED ON THIS REQUEST indicates that the specified STAE command was accepted for all load modules specified unless one or more of the load modules names are rejected on a subsequent DPP025I message.

zzz = THIS STAE REQUEST WILL NOT BE HONORED indicates that the STAE command option was neither "DUMP", "NODUMP", or "STEP".

Response: If the STAE option was invalid, select either "DUMP", "NODUMP", "ONEDUMP", or "STEP" option and re-issue the command.

DPP025I time LM NAME xxx - SPECIFIED ON STAE REQUEST IS INVALID AND WILL NOT BE PROCESSED

Routing code: 2

Message issued by segment: DPPTIMPS

Explanation: One of the load module names specified on the previous STAE command was found to be invalid. Message DPP024I was issued to notify the user of the option in effect as a result of that STAE command. Multiple DPP025I messages may be issued, one for each invalid load module name on that STAE command.

Response: Ensure that all characters in a specified load module name are either alphanumeric or one of the special characters "$", "#", "@". The special character "?" may also be used to delimit the load module name. The first character of a load module name cannot be numeric. Re-issue the STAE command with the corrected load module names.

DPP026I time INVALID IMP COMMAND

Routing code: 1

Message issued by segment: DPPXIMPP

Explanation: An invalid IMP command was issued.

Response: If the IMP command was misspelled, it should be reissued. If the specified IMP command was entered correctly, it is not known to the system. Therefore, the IMP command requested should be defined and added to the system.

DPP027I time OPERATOR COMMAND NOT DUMP OR NODUMP - THE SPECIAL REAL TIME OPERATING SYSTEM NOT CANCELLED

Routing Code: 2

Message issued by segment: DPPXKILL

Explanation: A cancel IMP command was issued which specified an action other than DUMP or NODUMP.

Response: A cancel IMP command should be issued which specifies an action of DUMP or NODUMP or the parameter should not be specified. If no parameter is specified, the IMP command parameter will default NODUMP.

DPP028I time TASK - DPPSAMP1 WAS ENTERED AT ENTRY POINT DPPSAMP1

Routing Code: 1

Message issued by segment: DPPSAMP1

Explanation: Test message issued by the Special Real Time Operating System sample program.

Response: None.

DPP029I time date RC = hhh cccccccccccccccccccc CONSOLE OS DESCRIPTOR AND ROUTING CODES xxxx ALTRC hhh

Routing Code: 1

Message issued by segment: DPPMMSGV

Explanation: The message displays the status (In or Out of Service) of system messages console routing codes. The message is issued in response to an MSGRC IMP command.

Response: None.

DPP030I time date RC = hhh cccccccccccccccccccc PROGRAM TASK NAME = tttttttt EPNAME = nnnnnnnn ALTRC = hhh

Routing code: 1

Message issued by segment: DPPMMSGV

Explanation: The message displays the status (In or Out of Service) of system messages output program routing codes. The message is issued in response to an MSGRC IMP command.

Response: None.

DPP031I time date RC = hhh cccccccccccccccccccc OS DEVICE DDNAME = dddddddd ALTRC = hhh

Routing code: 1

Message issued by segment: DPPMMSGV

Explanation: The message displays the status (In or Out-of-Service) of system messages OS DEVICE (printer, tape, etc. The message is issued in response to an MSGRC IMP command.

Response: None.

DPP032I time date RC = hhh cccccccccccccccccccc DISPLAY FUNC AREA = x ACCESS AREA = x ALTRC = hhh

Routing code: 1

Message issued by segment: DPPMMSGV

Explanation: The message displays the status (In or Out of Service) of system messages display routing codes. The message is issued in response to an MSGRC IMP command.

Response: None.

DPP033I time INVALID REQUEST - ALTERNATE ROUTE CODE IS OUT OF SERVICE

Routing Code: 1

Message issued by segment: DPPMMSGV

Explanation: An MSGRC IMP command was issued that specified an alternate route code that is out of service. An MSGRC IMP command should be issued that specifies no alternate route code or one that specifies an alternate route code that is in service. An MSGRC IMP command with a STATALL parameter can be specified to determine that route codes are in or out of service.

Response: None.

DPP034I time INVALID REQUEST - ROUTE CODE = ALTERNATE ROUTE CODE

Routing code: 1

Message issued by segment: DPPMMSGV

Explanation: An MSGRC IMP command was issued that specified the same route code in the primary and alternate route code parameters.

Response: An MSBRC IMP command should be issued that specifies different route codes for the primary and alternate route code parameters.

DPP035I time ROUTING CODE NOT FOUND OR ACTION (STATUS STATALL, IN OR OUT) PARAMETER NOT SPECIFIED

Routing code: 1

Message issued by segment: DPPMMSGV

Explanation: An MSGRC IMP command contains a route code not in the system or ACTION parameter, STATUS = STATALL-IN-OUT, not specified.

Response: The validity of the route code should be determined issuing an MSGRC IMP command with a STATALL parameter or a valid action parameter (STATUS-STATALL-IN-OUT) should be specified.

DPP036I DDSNAME = XXXXXXXX, COMPARE ENDED WITH I/O ERROR, RESULTS ON COMPRINT REPORT DATA SET

Routing code: 3

Message issued by segment: DPPSCMPR

Explanation: If IEBCOMPR returns with a return code G.7. 4, this message is output.

Response: None.

DPP037I USER DATA

Routing code: 2

Message issued by segment: None.

Explanation: This message is comprised of 50 characters of user data on which no translation is done. The data is output exactly as passed by the user. There is no use of this message by the released Special Real Time Operating System system.

Response: None.

DPP038I POSSIBLE SPECIAL REAL TIME OPERATING SYSTEM TIME ERROR - THE SPECIAL REAL TIME OPERATING SYSTEM TIME OF DAY HAS BEEN RECALCULATED

Routing code: 2

Message issued by segment: DPPCTIME or DPPCTIM2

Explanation: The time interval between successive updates to the Special Real Time Operating System time array exceeded the allowable tolerance, so a new time was calculated.

Response: None.

DPP039I THE OS/SPECIAL REAL TIME OPERATING SYSTEM TIME CONVERSION FACTOR HAS BEEN UPGRADED

Routing code: 2

Message issued by segment: DPPCUPCF

Explanation: A time correction value has been passed to the Special Real Time Operating System time management services and the Special Real Time Operating System time and date have been updated accordingly.

Response: None.

DPP041I xxx HAS BEGUN PROCESSING SYSINIT INPUT STREAM

Routing code: 2

Message issued by segment: DPPINIT1

Explanation: This message is used to notify the operator that the SYSINIT input stream is being processed. It is an indication that the Special Real Time Operating System has completed initialization. xxx - is used to identify the SYSINIT input stream.

xxx = "SLAVE JOB" indicates the SLAVE partition SYSINIT input streams.

xxx = "MASTER JOB" indicates the MASTER partition SYSINIT input stream.

xxx = "SRTOS JOB" indicates the realtime partition SYSINIT input stream in a single partition environment.

Response: None

DPP042I BLDL FAILED FOR MODULE MMMMMMMM - RETURN CODE WAS CCCCCCCC

Routing code: 2

Message issued by segment: DDDIDFIX

Explanation: The page fix routine had a load module fix request for module MMMMMMMM, a BLDL for the module failed, the return code was CCCCCCCC.

Response: Check array DPPXFIX and verify that the module to be fixed exists as a load module and the array is properly built.

DPP043I FIX FAILED FOR TTTTTTTTT - NNNNNNNN - RETURN CODE WAS CCCCCCCC

Routing code: 2

Message issued by segment: DPPIPFIX

Explanation: An attempt was made to fix virtual storage for TTTTTTTTT - (load module, named array, numbered array) - named NNNNNNNN - the return code from the page fix routine was CCCCCCCC.

Response: The contents of array DPPXFIX should be reviewed; the normal cause of this failure is too much real storage is becoming fixed.

DPP044I TASK = TTTTTTTT - EP=EEEEEEEE WAS POSTED WITH A NONZERO POST CODE - ECB CONTENTS = CCCCCCCC

Routing code: 2

Message issued by segment: DPINIT11

Explanation: Task named TTTTTTTT, with entry point named EEEEEEEE, was PATCHed by Special Real Time Operating System initialization. The task was PATCHed with the ECB= option and, at termination, the ECB had a non-zero completion code. The contents of the ECB were CCCCCCCC.

Response: Notify the responsible programmer.

DPP045I CONTROL STATEMENT ERROR DETECTED - RUN ABORTED

Routing code: 2

Message issued by segment: DPINIT05

Explanation: The SYSINIT input stream contained control statement(s) which were erroneous. The run is terminated.

Response: Have the erroneous control statements corrected and resubmit the run.

DPP046I MASTER OR SLAVE PARTITION WAITING FOR CORRESPONDING MASTER OR SLAVE TO BE INITIALIZED

Routing code: 1

Message issued by segment: DPINIT5

Explanation: A job has been started whose SYSINIT stream contained a MASTER or SLAVE statement. The job has reached a point in its initialization where it can go no further until its corresponding MASTER or SLAVE has been started. This message is issued at one-minute intervals until the corresponding MASTER or SLAVE job is initialized.

Response: Start the corresponding MASTER or SLAVE job.

DPP047I GETARRAY FAILED FOR DPPXFIX - RETURN CODE WAS CCCCCCCC - PAGE FFFFFFFF BYPASSED

Routing code: 2

Message issued by segment: DPPIPFIX and DPPDPFRE

Explanation: A PATCH to DPPIPFIX was issued; however, array DPPXFIX was not located. The return code from the GETARRAY for array DPPXFIX was CCCCCCCC. The Function (FIX or FREE) was The Function (FIX or FREE) was bypassed.

Response: Remove the PATCH to DPPIPFIX or be sure that array DPPXFIX exists in the data base being used (DBINIT and DBINIT2 cards).

DPP048I GETARRAY FAILED FOR TTTTTTTT - NNN - RETURN CODE WAS CCCCCCCC.

Routing code: 2

Message issued by segment: DPPIPFIX

Explanation: A PATCH was issued to DPPIPFIX. While processing array DPPXFIX, a fix request was found for TTTTTTTT (named array or numbered array), a GETARRAY was issued to locate the named or numbered array NNNNNNNN. The GETARRAY failed with return code CCCCCCCC.

Response: Check that the numbered or named array exists in the data base, or that array DPPXFIX is valid.

DPP049I ITEM IIIIIIII IN ARRAY DPPXFIX COULD NOT BE FIXED BECAUSE THE TYPE FIELD IS INVALID

Routing code: 2

Message issued by segment: DPPIPFIX

Explanation: A PATCH was issued to DPPIPFIX; while processing array DPPXFIX, an Item IIIIIIII was found that contained an invalid fix type.

Response: The only valid fix types are N, A and L. Have array DPPXFIX corrected.

DPP050I time date DRECOUT DD CARD MISSING

Routing code: 2

Message issued by segment: DPPXRINT

Explanation: Data recording initialization was unable to open the data recording DCB due to the absence of the DRECOUT DD statement in the job step JCL.

Response: A DRECOUT DD statement should be added to the job step JCL.

DPP051I time date DPBIN DD CARD MISSING

Routing code: 2

Message issued by segment: DPPXDPB

Explanation: Data playback was unable to open the data playback DCB due to the absence of the DPBIN DD statement from the job step JCL.

Response: A DPBIN DD statement should be added to the job step JCL.

DPP052I PAGE FIX FUNCTION COMPLETE - ALL ITEMS WERE XXXXXX

Routing code: 2

Message issued by segment: DPPIPFIX

Explanation: A PATCH was issued to DPPIPFIX. Array DPPXFIX was processed; when all processing had been completed, this message was issued to say all ITEMS were XXXXXX - (FIXED or NOT FIXED).

Response: If all ITEMS were not fixed, a decision may be required whether to continue or terminate this run.

DPP053I time date REPORT DATA OUTPUT FACILITY UNABLE TO OPEN SPECIFIED DATA SET

Routing code: 2

Message issued by segment: DPPXRPRT

Explanation: Report Data Output Facility unable to open specified DD statement due to the absence of the DD statement from the job step JCL.

Response: A DD statement should be added to the job step JCL for each DD name passed to the Report Date Output Facility or the DD name that is not defined by a DD statement should not be passed to the Report Data Output Facility.

DPP054I WRITING OF FAILOVER DATA SET BYPASSED FOR RESTART OF SLAVE PARTITION

Routing code: 2

Message issued by segment: DPPINIT1

Explanation: A MASTER and a SLAVE had been running, and the SLAVE terminated. The SLAVE is being restarted and its SYSINIT stream contains a RESTART WRITE statement. A failover data set had been written on the initial startup of the MASTER and SLAVE, so this function is bypassed.

Response: None.

DPP055I ERROR ON DDS DECLARATION

Routing code: 3

Message issued by segment: DPPSINIT

Explanation: One of the DDSNAMES cards has been incorrectly coded (within the DDSCTLIN stream).

Response: Correct the bad DDSNAMES card and resubmit the job.

DPP056I DDSNAME = XXXXXXXX, PRIMARY = YYYYYYYY, BACKUP = ZZZZZZZZ (= OUT-OF-SERVICE)

Routing code: 3

Message issued by segment: DPPSMSGI

Explanation: This message only gives the status of a specified DDS.

Response: None.

DPP057I DDSNAME = XXXXXXXX IS LOCKED OUT

Routing code: 3

Message issued by segment: DPPSLOCK

Explanation: A DDS LOCK is being placed on the DDS in question.

Response: None.

DPP058I DDSNAME = XXXXXXXX IS UNLOCKED

Routing code: 3

Message issued by segment: DPPSUNLK

Explanation: A DDS lock is being taken off the DDS in question.

Response: None.

DPP059I UNABLE TO CREATE BACKUP FOR DDSNAME = XXXXXXXX

Routing code: 3

Message issued by segment: DPPSCRBK

Explanation: An attempt to create a backup copy is unsuccessful because of I/0 errors.

Response: None.

DPP060I time date (up to 80 characters of user data)

Routing code: 2

Message issued by segment: DPPXKILL

Explanation: The Special Real Time Operating System cancel routine displays any operator comments, passed to the cancel routine, about the nature of the termination.

Response: None.

DPP061I PTIME for TASK X EP Y TERMINATED BY PATCH RC Z

Routing code: 2

Message issued by segment: DPPCPTIM

Explanation: A PATCH was issued by a time service routine in response to a previous PTIME request to task X and EP Y, but received an error return code of Z from the PATCH routine. Therefore, its PTIME for this task and EP have been deleted. The return code is output as a hexadecimal value.

Response: None.

DPP062I DDS REQUEST REJECTED

Routing code: 3

Message issued by segment: DPPSMSGI

Explanation: A switch was attempted while the backup was out of service.

Response: None.

DPP063I DDSNAME = XXXXXXXX WAS NOT DECLARED DURING INITIALIZATION

Routing code: 3

Message issued by segment: DPPSMSGI

Explanation: This message follows DPP062I stating that the DDSNAME specified is a user command that could not be found among the ones declared as duplicates at iniatization time.

Response: None.

DPP064I DDSNAME = XXXXXXXX, BACKUP IS ALREADY IN SERVICE

Routing code: 3

Message issued by segment: DPPSCRBK

Explanation: The backup is already in service, so a user request to create a backup will not be executed.

Response: None.

DPP065I DDSNAME = XXXXXXXX = S CURRENTLY OPENED BY ANOTHER TASK

Routing code: 3

Message issued by segment: DPPSMSGI

Explanation: The user's request for 'REPLACE' cannot be satisfied because the DDS is already open.

Response: None.

DPP066I time date

Routing code: 1

Message issued by segment: DPPZSAMP DPPSAMP1

Explanation: Test message issued by the Special Real Time Operating System sample program.

Response: None.

DPP067I ABEND sssuuu AT LOCATION xxxxxx DURING THE SPECIAL REAL TIME OPERATING SYSTEM SERVICE OF PL/I - FORT MACRO ID yy

Routing code: 3

Message issued by segment: DPPPIF

Explanation: The message is intended to inform the high level language user that a Special Real Time Operating System service routine ABENDed with a completion code of sssuuu where sss is the system completion code and uuu is the user completion code at location xxxxxx for service call identified by MACRO ID yy.

Response: None.

DPP068I time date #8C1# MACRO FUNCTIONING

Routing code: 1

Message issued by segment: DPPZSAMP

Explanation: Test message issued by the Special Real Time Operation System sample program. When the sample program executes a macro, and the macro executed properly, message DPP068 will be issued with the macro name.

Response: None.

DPP069I time date ITEM DPPSAMP2 CONTENTS ARE #6C1#

Routing code: 1

Message issued by segment: DPPZSAMP

Explanation: Test message issued by the Special Real Time Operating System sample program. A GETITEM macro will be executed by the sample program. The contents of item DPPSAMP2 (in array DPPZSAMP) will be displayed via message DPP069.

Response: None.

DPP070I time (content of the IMP command which was received)

Routing code: 1

Message issued by segment: DPPXIMPP

Explanation: Contains the IMP command issued by the operator.

Response: None.

DPP071I DDS REQUEST REJECTED - REQUEST NOT UNDERSTOOD

Routing code: 3

Message issued by segment: DPPSMSGI

Explanation: A DDS request was entered with a bad format, and the DSS input handler could not interpret it.

Response: None.

DPP072I time date PROGRAM #8C1# PATCHED AS A RESULT OF AN IMP COMMAND APPEARS TO BE IN A LOOP

Routing code: 2

Message issued by segment: DPPXIMPP

Explanation: An IMP command was issued to the SLAVE partition and the routine that processes (routine patched by IMP as a result of this command) the command appears to be in a loop.

Response: The loop in the processing routine for this particular IMP command should be corrected. The loop will not affect the operation of the Special Real Time Operating System.

DPP073I DDSNAME = XXXXXXXX, UNABLE TO ACCESS DATA SETS COMPARE REJECTED

Routing code: 3

Message issued by segment: DPPSCMPR

Explanation: A DDS compare request is being rejected because the JFCBs for the data sets cannot be read.

Response: None.

DPP074I DDSNAME = XXXXXXXX, DATA SETS NOT SAME TYPE COMPARE REJECTED

Routing code: 3

Message issued by segment: DPPSCMPR

Explanation: A DDS compare request was made for data sets with different DSORG fields in the DSCBS, so the request is being dropped.

Response: None.

DPP075I DDSNAME = XXXXXXXX, COMPARE IN PROGRESS

Routing code: 3

Message issued by segment: DPPSCMPR

Explanation: The DDS specified is now being compared.

Response: None.

DPP076I DDSNAME = XXXXXXXX, COMPARE ENDED DATA SETS ARE EQUAL

Routing code: 3

Message issued by segment: DPPSCMPR

Explanation: A DDS compare function ended, and the two data sets were found to be equal.

Response: None.

DPP077I DDSNAME = XXXXXXXX, COMPARE ENDED, DATA SETS ARE NOT EQUAL

Routing code: 3

Message issued by segment: DPPSCMPR

Explanation: A DDS compare function ended with the data sets not being equal; IEBCOMPR output is on the COMPRINT report data sets.

Response: None.

DPP078I DDSNAME = XXXXXXXX, COMPARE REJECTED, NO //DDSCMPIN DD CARD FOR SYSIN

Routing code: 3

Message issued by segment: DPPSCMPR

Explanation: A DDS compare request is being dropped because there is no DDSCMPIN DD card in the II07 to hold IEB COMPR input.

Response: None

DPP079I time IMP PARAMETERS EXCEED MAXIMUM PARAMETERS DEFINED FOR IMP COMMAND cccccccc

Routing code: 1

Message issued by segment: DPPXIMPP

Explanation: Too many parameters were passed by the IMP command.

Response: The IMP command should be issued again, not exceeding the maximum number of parameters for this particular IMP command.

DPP080A FAIL/RST DATA SET NOT WRITTEN - END OF EXTENT ON DPPFAIL

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: The data set named in the DPPFAIL DD card contains insufficient space for the failure/restart data set.

Response: Allocate more space.

DPP081A FAIL/RST DATA SET NOT WRITTEN - I/O ERROR ON DPPFAIL

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: An I/O error occurred on the Failure/Restart data set.

Response: Use a different disk or allocate the space at a different place on the disk pack. Hardware error.

DPP082A FAIL/RST DATA SET NOT WRITTEN - I/O ERROR READING PAGING DATA SET

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: An I/O error reading the OS/VS1 paging data set.

Response: An IPL will be required. Hardware error.

DPP083A FAIL/RST DATA SET WRITTEN - I/O READING JOBQUEUE/SYSWADS

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: An I/O error occurred while reading the OS/VS1 Job queue or SYS1.SYSWADS data sets.

Response: An IPL will be required. Hardware error.

DPP084A FAIL/RST DATA SET NOT WRITTEN - I/O ERROR READING SWADS

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: An I/O error occurred while reading the SWADS for the MASTER partition.

Response: Hardware error. A different SWADS will probably be required.

DPP085A FAIL/RST DATA SET NOT WRITTEN - I/O READING SWADS FOR SLAVEPART

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: An I/O error occurred while reading the SWADS for the SLAVE partition.

Response: Hardware error. A different SWADS will probably be required.

DPP086A FAIL/RST DATA SET NOT WRITTEN - PROG CK. IN RESTART WRITE

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: An unexplained program check occurred in restart write.

Response: Probable programming error in Failure/Restart.

DPP087A FAIL/RST DATA SET NOT WRITTEN - MACHINE CHECK IN RESTART

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: A hardware error occurred in restart write.

Response: Retry the job.

DPP088A SECNDRY COPY OF FAIL/RST DATA SET NOT WRITTEN I/O ERROR ddname

Routing code: 5

Message issued by segment: DOMIRCPY

Explanation: An I/O error occurred while attempting to make backup copies of the failure/restart data set. No backup copies were made. Insufficient space in the data set can cause this error.

Response: Possible hardware error. Allocate the backup failure/restart data set at a different location or increase its size.

DPP089A DDNAME ddname INVALID FOR COPY OF F/R DATA SET

Routing code: 5

Message issued by segment: DOMIRCPY

Explanation: The ddname indicated is invalid for the failure/restart data set for one or more of the following reasons:

- It is not a direct access device of the same type as the primary F/R data set.
- Another F/R data set is on the volume.
- The volume contains the SYS1.NUCLEUS data set. No backup copies were made.

Response: Correct the JCL.

DPP090I FAIL/RST DATA SET WRITTEN

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: The Failure/Restart data set has been successfully written.

Response: None.

DPP091I FAIL/RST DATA SET READ COMPLETE

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: The Failure/Restart data set had been successfully IPLed.

Response: None.

DPP092A FAIL/RST DATA SET NOT WRITTEN - DPPFAIL DD CARD INVALID OR MISSING

Routing code: 5

Message issued by segment: DOMIRFL2

Explanation: The Failure/Restart data set was not written because of one or more of the following:

- No DPPFAIL DD card is provided.
- The data set name in the DPPFAIL DD card is not on direct access.
- The data set named in the DPPFAIL DD card is on the same volume with SYS1.NUCLEUS.

Response: Correct the JCL and resume the job.

DPP093I FAIL/RST DATA SET NOT WRITTEN - OTHER R/T JOB IN SYSTEM

Routing code: 5

Message issued by segment - DOMIRFL2

Explanation: Another realtime job in the same OS/VS1 system owns restart write eligibility.

Response: None.

DPP094A PROBE FUNCTION NOT RUNNING IN OTHER CPU

Routing code: 5

Message issued by segment: DOMIRCMN

Explanation: This message can appear only in systems with CMCKPRB=YES specified in the FAILRST SYSGEN macro. It indicates that neither a continuous monitor or a PROBE is running in the backup CPU.

Response: Start a PROBE function in the backup CPU if desired.

DPP095I PROBE FUNCTION IS NOW RUNNING IN OTHER CPU

Routing code: 5

Message issued by segment: DOMIRCMN

Explanation: This message can appear only in systems with CMCKPRB=YES specified in the FAILRST SYSTEN macro. It indicates that the continuous monitor has detected that a PROBE function is running in the other CPU.

Response: None.

DPP096I ANOTHER CONT. MON IS IN OTHER CPU

Routing code: 5

Message issued by segment: DOMIRCMN

Explanation: This message can appear only in systems with CMCKPRB=YES specified in the FAILRST SYSGEN macros. It indicates that each CPU has a continuous monitor running in duplex mode. Each CPU is operating as though it were the prime CPU.

Response: Cancel the realtime job in one of the CPUs unless the configuration is specified.

DPP098A CONTINUOUS MONITOR RECOMMENDS FAILOVER

Routing code: 5

Message issued by segment: DOMIRCMN

Explanation: The continuous monitor has detected an error in the online system and is recommending a failure or restart.

Response: Allow the failover to occur or invoke a restart as appropriate.

DPP099I ANOTHER CONT. MON/PROBE ON SAME SYSTEM - NO HARDWARE FAILOVER RECOM BY THIS CONT. MON

Routing code: 5

Message issued by segment: DOMIRCMN

Explanation: One of the following conditions exists:

- This realtime job does not own restart write eligibility.
- A PROBE function is running in another job on this CPU.
- A continuous monitor is already running in duplex mode on this CPU.

Response: None.

DPP800I CONTROL STATEMENT LABEL MUST NOT EXCEED EIGHT CHARACTERS

Routing code: SYSPRINT

Message issued by segment: DPPINITO

Explanation: The LABEL field of a control statement had a name that exceeded eight characters. Eight is the maximum allowable number of characters in this field.

Response: Correct the name in the LABEL field and resubmit.

DPP801I CONTROL STATEMENT MUST HAVE OPERANDS

Routing code: SYSPRINT

Message issued by segment: DPPINITO,DPINITOA

Explanation: All control statements except ABEND must have OPERANDS that begin on the same card as the OPERATION.

Response: Supply the proper OPERANDS and resubmit.

DPP802I INVALID OPERATION FIELD

Routing code: SYSPRINT

Message issued by segment: DPPINITO

Explanation: An OPERAND other than one of the following was found: PATCH, WAIT, WRITE, TCB, GETWA, CBGET, ABEND, MASTER, SLAVE, DBREF.

Response: Correct the OPERATION field and resubmit.

DPP803I TOO MANY OPERANDS ON CONTROL STATEMENT

Routing code: SYSPRINT

Message issued by segment: DPINIT03

Explanation: The maximum number of OPERAND characters allowed on any one control statement and its continuations is 255 characters.

Response: Correct the control statement and resubmit.

DPP804I INVALID CONTROL STATEMENT CONTINUATION

Routing code: SYSPRINT

Message issued by segment: DPINIT03

Explanation: A continuation was indicated and the card processed either began in column 15 or before, or processing was not within quotes and the continuation did not start in column 16.

Response: Correct the control statement and retry.

DPP805I INVALID OPERAND ON CONTROL STATEMENT

Routing code: SYSPRINT

Message issued by segment: DPINIT04,DPINITOA

Explanation: The control statement contains an invalid operand for the operation type specified in the operation field.

Response: Correct the control statement and retry.

DPP806I ONLY ONE MASTER OR SLAVE STATEMENT ALLOWED IN INPUT STREAM

Routing code: SYSPRINT

Message issued by segment: DPPINITO

Explanation: Only one MASTER or SLAVE control statement is allowed in each SYSINIT input stream.

Response: Remove the extra MASTER or SLAVE statement(s) from the stream and retry.

DPP807I NAME SPECIFIED ON WAIT STATEMENT NOT A LABEL ON A PREVIOUS PATCH CONTROL STATEMENT

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The operand on the WAIT statement was not a label from a PATCH statement which precedes the WAIT in the input stream.

Response: Correct the WAIT operand, remove the WAIT, or place the proper PATCH statement ahead of the WAIT in the input stream and retry.

DPP808I ONLY ONE RESTART STATEMENT ALLOWED IN THE INPUT STREAM

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The input stream contained more than one WRITE RESTART statement. Only one is valid.

Response: Remove the extra WRITE RESTART statement(s) from the SYSINIT stream and retry.

DPP809I INVALID NAME IN EP=FIELD

Routing code: SYSPRINT

Message issued by segment: DPINIT02

Explanation: The name specified on an EP= keyword of a PATCH statement exceeded eight characters.

Response: Correct the EP= operand and retry.

DPP810I INVALID NAME IN TASK= FIELD

Routing code: SYSPRINT

Message issued by segment: DPINIT02

Explanation: The TASK= keyword of a PATCH statement contained a task name which exceeded eight characters.

Response: Correct the TASK= operand and retry.

DPP811I QL FIELD INVALID

Routing code: SYSPRINT

Message issued by segment: DPINIT02,DPPINITOA

Explanation: The QL= keyword on a PATCH statement contained a value of greater than 999.

Response: Correct the QL= keyword operand and retry.

DPP812I ID FIELD INVALID

Routing code: SYSPRINT

Message issued by segment: DPINIT02

Explanation: The ID= keyword on a PATCH statement contained an ID greater than 255.

Response: Correct the ID= keyword operand and retry.

DPP813I INVALID KEYWORD

Routing code: SYSPRINT

Message issued by segment: DPPINITO,DPINITOA

Explanation: The control statement contained an invalid keyword for the operation type specified in the operation field.

Response: Correct the keyword and retry.

DPP814I PRTY REFERENCE VALUE MISSING OR INVALID

Routing code: SYSPRINT

Message issued by segment: DPINIT02,DPINITOA

Explanation: The PRTY= keyword on the PATCH statement was either missing the reference value or the value exceeded 255.

Response: Correct the PRTY reference value and retry.

DPP815I INVALID DELIMETER IN PRTY OPERAND

Routing code: SYSPRINT

Message issued by segment: DPINIT02

Explanation: The PRTY= keyword on a PATCH statement was not coded as JOBSTEP - or (jobname,). The delimiter must be the minus (-) or the comma (,).

Response: Correct the PRTY= keyword operand and ensure that if (jobname,) is used that the specified jobname does not exceed eight characters.

DPP816I INVALID TASK NAME IN PRTY REFERENCE FIELD

Routing code: SYSPRINT

Message issued by segment: DPINIT02

Explanation: The PRTY= keyword on the PATCH statement contained a name in the (jobname,) field which exceeded eight characters.

Response: Correct the PRTY reference jobname and resubmit.

DPP817I DUPLICATE KEYWORD ON PATCH STATEMENT

Routing code: SYSPRINT

Message issued by segment: DPINIT02

Explanation: A keyword operand on the PATCH statement appeared twice on a single control statement.

Response: Correct the control statement and retry.

DPP818I INVALID DELIMETER IN PARAM SUBOPERANDS

Routing code: SYSPRINT

Message issued by segment: DPINIT01

Explanation: All suboperands within a PARAM field must end with a single quote (') character and be delimited by a comma. This PATCH statement contained a suboperand that was not delimited with a comma.

Response: Correct the control statement and retry.

DPP819I INVALID DATA IN PARAM FIELD

Routing code: SYSPRINT

Message issued by segment: DPINIT01

Explanation: The PARAM= keyword operand on a PATCH statement contained non-decimal data in an F' ' field, or non-hexadecimal data in an X' ' field.

Response: Correct the PARAM data and retry.

DPP820I INVALID DATA TYPE IDENTIFIER IN PARAM FIELD - MUST BE X, F, OR C.

Routing code: SYSPRINT

Message issued by segment: DPINIT01.

Explanation: The PARAM= keyword on a PATCH statement contains a data type identifier other than X, F, or C.

Response: Correct the data type identifier and retry.

DPP821I CHARACTER FOLLOWING PARAM DATA TYPE IDENTIFIER MUST BE A QUOTE

Routing code: SYSPRINT

Message issued by segment: DPINIT01

Explanation: The data type identifier (F, C, or X) on a PATCH statement must be followed by a single quote (') character.

Response: Correct the PARAM field and retry.

DPP822I UNBALANCED QUOTES IN PARAM FIELD

Routing code: SYSPRINT

Message issued by segment: DPINIT01

Explanation: A PARAM keyword on a PATCH statement must contain evenly balanced single quote (') characters. This character is not valid with a PARAM suboperand.

Response: Correct the PARAM statement and retry.

DPP823I PARAM FIELD MUST END WITH RIGHT PARENTHESIS

Routing code: SYSPRINT

Message issued by segment: DPINIT01

Explanation: The PARAM= keyword suboperands on a PATCH statement must be enclosed in parentheses (--); the ending or right parenthesis is missing on this PATCH statement.

Operator response: Correct the PARAM field and retry.

DPP824I PARAM FIELD MUST START WITH LEFT PARENTHESIS

Routing code: SYSPRINT

Message issued by segment: DPINIT01

Explanation: The PARAM= keyword suboperands on a PATCH control statement must be enclosed in parentheses (--); the beginning or left parenthesis on this PATCH statement is missing.

Response: Correct the PARAM field and retry.

DPP825I INVALID DELIMETER FOLLOWING PARAM OPERAND

Routing code: SYSPRINT

Message issued by segment: DPINIT01

Explanation: All operands on a PATCH statement must be delimited with a comma or blank. This control statement has a character other than a comma or blank following the ending (right) parenthesis.

Response: Correct the statement and retry.

DPP826I QL FIELD CONTAINS NONDECIMAL DATA

Routing code: SYSPRINT

Message issued by segment: DPINIT02

Explanation: The QL= operand on the PATCH statement contained a value that included non-decimal characters.

Response: Correct the QL= operand and retry.

DPP827I ID FIELD CONTAINS NONDECIMAL DATA

Routing code: SYSPRINT

Message issued by segment: DPINIT02

Explanation: The ID= keyword on the PATCH statement contained a value that included non-decimal data.

Response: Correct the ID field and retry.

DPP828I PRTY FIELD CONTAINS NONDECIMAL DATA

Routing code: SYSPRINT

Message issued by segment: DPINIT02,DPINITOA

Explanation: The PRTY = keyword on the PATCH statement contained a priority reference value that included a non-decimal character(s).

Response: Correct the PRTY reference value and retry.

DPP829I CBGET DATA IS NONDECIMAL

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The CBGET statement operand contained a value which included a non-decimal character(s).

Response: Correct the CBGET operand and retry.

DPP830I INVALID JOBNAME

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: A MASTER or SLAVE statement had an invalid jobname on its operand. The jobname exceeds eight characters.

Response: Correct the MASTER or SLAVE statement jobname and retry.

DPP831I TIME FIELD CONTAINS NONDECIMAL DATA

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The time field on the ABEND card contained a value that included a non-decimal character(s).

Response: Correct the time field on the CBGET statement and retry.

DPP832I TIME FIELD CANNOT BE GREATER THAN 999

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The time field on the ABEND card contained a value greater than 999.

Response: Correct the time field and retry.

DPP833I SECOND OPERAND ON AN ABEND STATEMENT MUST BE DUMP OR OMITTED

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The second operand on the ABEND statement contained characters other than the word 'DUMP'. This operand must be 'DUMP' or omitted.

Operator response: Correct the ABEND statement and retry.

DPP834I NUMBER OF TCBS IS NONDECIMAL

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The operand on the TCB statement contained a value that included a non-decimal character(s).

Response: Correct the TCB statement and retry.

DPP835I EP= MUST BE SPECIFIED ON A PATCH STATEMENT

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The PATCH statement processed did not have the EP= keyword specified.

Response: Correct the EP= and retry.

DPP836I INPUT DCB - SYSINIT - FAILED TO OPEN

Routing code: SYSPRINT

Message issued by segment: DPPINITO

Explanation: The input DCB for the SYSINIT data set could not be opened.

Response: Check the SYSINIT DD card and verify that it allocates the correct data set.

DPP837I INVALID SUBPARAMETERS IN LIST

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The GETWA statement contained subparameters in its size list which were invalid or had invalid delimiters.

Response: Correct the GETWA statement and retry.

DPP838I LIST ENTRY CONTAINS NONDECIMAL DATA

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The GETWA suboperand list contained subparameters that contained a non-decimal character(s).

Response: Correct the GETWA statement and retry.

DPP839I NUMBER OF BLOCKS CANNOT EXCEED 4095

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The GETWA statement had a request for a number of blocks and the request exceeded the maximum of 4095.

Response: Correct the GETWA statement and retry.

DPP840I GETWA SIZE EXCEEDS 30720 OR GREATER THAN 2048 AND NOT A 2K MULTIPLE OR NOT A MULTIPLE OF 8

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The GETWA statement contained a request for a block size that exceeded the maximum size or is an invalid size.

Response: Correct the GETWA statement and continue.

DPP841I EXCESSIVE NUMBER OF SUBOPERANDS IN LIST -64 IS THE MAXIMUM

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The GETWA statement contained a list of subparameters with more than 64 subparameters.

Response: Correct the GETWA statement and retry.

DPP842I SUBLIST MUST END WITH RIGHT PARENTHESIS

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: the GETWA statement contained a list of subparameters that did not end with a right parenthesis.

Response: Correct the GETWA statement and retry.

DPP843I SUBLIST MUST BEGIN WITH A LEFT PARENTHESIS

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The GETWA statement contained a subparameter list that did not begin with a left parenthesis.

Response: Correct the GETWA statement and retry.

DPP844I CONTINUATION EXPECTED - NOT RECEIVED

Routing code: SYSPRINT

Message issued by segment: DPINIT05

Explanation: The last statement read from the input stream indicated that a continuation statement was to follow. The continuation statement was not in the input stream.

Response: Correct the last statement or add the continuation statement and retry.

DPP845I TWO-PARTITION FUNCTION NOT AVAILABLE

Routing code: SYSPRINT

Message issued by segment: DPINIT04

Explanation: The input stream contains a MASTER or SLAVE statement; however, at SRTOS SYSGEN, two-partition operation was not selected.

Response: Remove the MASTER or SLAVE statement and retry.

DPP846I nnnnnnnn DEFINED AS QUEUE HOLDER BUT NOT REFERENCED BY ANY QUEUE PROCESSOR

Routing code: SYSPRINT

Message issued by segment: DPINIT05

Explanation: A QH statement defined nnnnnnnn as a queue holder but that name is not specified on any QP statement. If allowed to go into execution, work that is queued to this queue holder could never be executed.

Response: Remove the QH card that specified this name or add this name to some QP statement and retry.

DPP847I nnnnnnnn REFERENCED AS QUEUE HOLDER BY A QUEUE PROCESSOR BUT NOT DEFINED

Routine Code: SYSPRINT

Message issued by segment: DPINIT05

Explanation: The name nnnnnnnn appears in the QH= operand of a QP statement but is not defined as a queue holder by a QH statement.

Response: Define a queue holder by this name or delete this name from the QP statement that references it and retry.

DPP848I {QH NAME}

Routing code: SYSPRINT

Message issued by segment: DPINIT0A

Explanation: A QH name or QP number is required on every QH or QP statement as a positional parameter. This parameter is missing or invalid on the user statement preceding this message.

Response: Correct the statement and retry.

DPP849I cccccccc OPERAND CONTAINS TOO MANY, TOO FEW OR ILLEGAL CHARACTERS IN PARAMETER OR SUB-PARAMETER

Routing code: SYSPRINT

Message issued by segment: DPINIT0A

Expalanation: The operand specified by cccccccc is invalid on the user statement preceding this message.

Response: Correct the statement and retry.

DPP850A SxxIN TASK yyyyyyyy. ABEND #nnn FOR MODULE zzzzzzzz. REPLY 'YES' TO ALLOW DUMP OR 'NO' TO SUPPRESS DUMP.

Routing code: The message is issued as an OS/VS1 WTOR

Message issued by segment: DPPSTAE

Explanation: A subtask ABENDed while the "STAE.OPTION" request was in effect. The operator may reply 'YES' to allow the dump to be formatted or 'NO' to suppress the dump. If the operator has not replied to this WTOR in 5 minutes, the WTOR will be cancelled and the dump formatting will be bypassed. The message defines the type of ABEND and the module responsible, where:

XXX - is the ABEND code

YYYYYYYY - is the task name

nnn - is the total number of ABENDs for this module

ZZZZZZZZ - is the entry point name of the module

Messages DPP860 and DPP861 are issued in conjunction with this WTOR to provide the PSW and register contents at the time of the ABEND.

Response: Reply 'YES' to allow the dump to be formatted. Reply 'NO' to suppress formatting of the dump.

DPP851I

```
{YES}
{NO } DUMP REPLY ACCEPTED
REPLY NOT RECEIVED IN TIME INTERVAL.  DUMP BYPASSED
XXX IS AN INVALID REPLY
```

Routing code: The message is issued as an OS/VS1 WTO

Message issued by segment: DPPTSTAE

Explanation: This message is issued in response to the operator reply to WTOR DPP850A. 'DPP851I 'YES' DUMP REPLY ACCEPTED' indicates that the operator reply was valid and issued within the time interval and a dump will be formatted. 'DPP851I 'NO' DUMP REPLY ACCEPTED' indicates that the operator reply was valid and issued within the time interval and dump formatting will be bypassed. 'DPP851I REPLY NOT RECEIVED IN TIME INTERVAL. DUMP BYPASSED' indicates that no operator reply was received within the time interval and dump formatting will be bypassed. 'DPP851I XXX IS AN INVALID REPLY' states that the operator reply was not a 'YES' or 'NO' and the WTOR DPP850A will be reissued.

Response: None.

DPP852I cccccccc OPERAND CONTAINS INVALID DATA

Routing code: SYSPRINT

Message issued by segment: DPINITOA

Explanation: The operand specified by ccccccc on the user statement preceding this message contains invalid data.

Response: Correct the statement and retry.

DPP853I cccccccc OPERAND DATA MUST BE ENCLOSED IN PARENTHESIS. ONE OR BOTH ARE MISSING

Routing code: SYSPRINT

Message issued by segment: DPINITOA

Explanation: The operand data of the parameter specified by cccccccc must be enclosed in parenthesis. Either the opening or closing parenthesis or both are missing on the user statement preceding this message.

Response: Correct the statement and retry.

DPP854I cccccccc

Routing code: SYSPRINT

Message issued by segment: DPINITOA

Explanation: The operand specified by cccccccc is required, but not correctly provided on the user statement preceding this message. Other messages may appear in conjunction with this message.

Response: Correct the statement and retry.

DPP855I cccccccc SPECIFIED AS EXIT= OPERAND NOT FOUND ON STEPLIB/JOBLIB DATA SET

Routing code: SYSPRINT

Message issued by segment: DPINITOA

Explanation: A STAEX command specifies cccccccc as an exit routine load module. The initialization routine has executed a BLDL and found that the load module could not be fetched if it should be needed.

Response: Add a load module by the specified name to the STEPLIB/JOBLIB data set(s) and retry.

DPP856I {QP NUMBER}
{QH NAME } SPECIFIED ON THIS STATEMENT HAS BEEN SPECIFIED ON A PREVIOUS STATEMENT

Routing code: SYSPRINT

Message issued by segment: DPINITOA

Explanation: The queue processor number or queue holder name specified as a positional parameter on the user statement preceding this message has been defined on a previous QP or QH statement.

Response: Remove the duplicate specifications and retry.

DPP857I cccccccc IS CONNECTED TO MORE THAN 21 OTHER BLOCKS

Routine code: SYSPRINT

Message issued by segment: DPINITOA

Explanation: the QH= operand of a QP statement is not allowed to contain more than 21 queue holder names and a queue holder name is not allowed to appear in more than 21 QP statement QP= operands. cccccccc is the QH or QP name that violates this restriction. A QP name is in the format ****QPnn, where nn is the user defined queue processor number.

Response: Reduce the number of references to the specified name and retry.

DPP860 PSW AT ABEND xxxx xxxx

Routing code: 1

Message issued by segment: DPPTSTAE

Explanation: Whenever an OS dump is suppressed by the STAE option processing (i.e., "STAE,NODUMP") or whenever "STAE,OPTION" is in effect, messages DPP860 and DPP861 are issued to provide a mini dump. Message DPP860 provides the PSW at the time of the ABEND.

Response: None.

DPP861 REGS aaaa bbbb cccc dddd

Routine code: 1

Message issued by segment: DPPTSTAE

Explanation: Whenever an OS dump is suppressed by the STAE option processing (i.e., "STAE,NODUMP") or whenever "STAE,OPTION" is in effect, messages DPP860 and DPP861 are issued to provide a mini dump. Message DPP861 is issued four times to provide the contents of registers 0-3, 4-7, 8-11, and 12-15, respectively at the time of the dump.

Response: None.

```
                                     {QP}     {PATCH}
DPP862I            QQQQQQQQ:   IS    {QH} ,   {NOPATCH}
                                     {TSK}
                   {HOLD}  {SEQ}
                   {REL} , {NONSEQ},CQL=nnn,[A]
```

Routine code: 2

Message issued by segment: DPPTQIMP

Explanation: This message is output as a result of the entry of every QS command. It reports the status of the queue processor(s), queue holder(s), and/or independent task(s) specified in the QS command.

QQQQQQQQ is the name of the unit being reported

QP - This is a queue processor

QH - This is a queue holder

TSK - This is an independent task

PATCH - This unit is allowed to accept work (PATCHes)

NOPATCH - PATCHes to this unit will be rejected

HOLD - This unit is not allowed to start processing any new work

REL - This unit can process any work which it is eligible to process

SEQ - (meaningful for QH only) only one QP may be processing work from this QH

NONSEQ - (meaningful for QH only) any QP connected to this QH may take work from this QH

CQL=nnn - nnn is the number of work queues currently awaiting processing

A - If present, this unit (TSK or QP only) is currently processing a piece of work

Response: None.

```
                          {QH}
DPP863I        QQQQQQQQ IS {QP} XREF TO:
               nnnnnnnn,...,nnn nnn
```

Routing code: 2

Message issued by segment: DPPTQIMP

Explanation: This message is output as a result of the entry of a QS command with the SREF operand. It is output following message DPP862. QQQQQQQQ is the unit being reported. QP or QH specifies the type of unit, queue processor or queue holder. If QP, the names following (nnnnnnnn,...) are the queue holders from which this QP may select work. If QH, the names following are the queue processors that may select work from this QH. Up to 7 names of connected units may appear in each message. Up to 3 messages may be output to output all connections to one unit.

Response: None.

DPP864I QS COMMAND PARAMETER PPPPPPPP INVALID. COMMAND IGNORED

Routing code: 2

Message issued by segment: DPPTQIMP

Explanation: A QS IMP command was entered with a misspelled, out of sequence or invalid parameter. The unacceptable parameter is reproduced as PPPPPPPP.

Response: Reenter request with correct parameters.

DPP865 COPY FAILED FOR 'FROM-00' TO 'TO-00'

Routing code: 3

Message issued by segment: DPPSRTCP

Explanation: The realtime copy operation pursuant to an RTCOPY command has failed.

Response: None.

DPP866 UNABLE TO READ UFCBS FOR '00NAME'

Routine code: 3

Message issued by segment: DPPSRTCP

Explanation: In a realtime copy operation, the JFCB for the DDname could not be read.

Response: None.

DPP867 UNABLE TO READ COUNT FOR 'DDNAME'

Routing code: 3

Message issued by segment: DPPSRTCP

Explanation: In a realtime copy operation the count fields for ddname could not be read.

Response: None.

DPP868 UNABLE TO READ RO FOR ddname

Routine code: 3

Message issued by segment: DPPSRTCP

Explanation: In a realtime copy operation, RO could not be read for ddname.

Response: None.

DPP869 UNABLE TO READ DATA FOR ddname

Routing code: 3

Message issued by segment: DPPSRTCP

Explanation; In a realtime copy operation, the data fields could not be read for ddname.

Response: None.

DPP870 UNABLE TO WRITE DATA FOR DDNAME

Routine code: 3

Message issued by segment: DPPSRTCP

Explanation: In a copy operation, the data could not be written for ddname.

Response: None.

DPP871 COPY ENDED FOR 'ddname-1' TO 'ddname-2'

Routing code: 3

Message issued by segment: DPPSRTCP

Explanation: The realtime copy operation requested by a RTCOPY command for ddname-1 to ddname-2 has ended.

Response: None.

DPP880 UNABLE TO OPEN DDSTATUS, RUNNING SINGLE MODE

Routine code: 3

Message issued by segment: DPPSINIT

Explanation: When attempting to run REFRESH or READONLY mode, the DDSTATUS data set could not be opened.

Response: None.

DPP881 UNABLE TO WRITE DDSTATUS RECORD

Routine code: 3

Message issued by segment: DPPSWRST

Explanation: The status of a DDS changed (or was being initialized) but the record could not be written to the DDSTATUS data set.

Response: None.

DPP882 UNABLE TO READ DDSTATUS RECORD, RUNNING SINGLE MODE

Routing code: 3

Message issued by segment: DPPSINIT

Explanation: When running REFRESH or READONLY, the DDSTATUS data set record could not be read.

Response: None.

DPP883 DDSTATUS NOT OPEN FOR OUTPUT

Routing code: 3

Message issued by segment: DPPSWRST

Explanation: The status of a DDS changed (or was being initialized) but the DDSTATUS data set could not be opened for output.

Response: None.

DPP884 DDSTATUS NOT UPDATED, RUNNING IN READONLY MODE

Routing code: 3

Message issued by segment: DPPSWRST

Explanation: The status of a DDS changed while running in READONLY mode, so the DDSTATUS data set will not be updated.

Response: None.

DPP885 DDSTATUS HAS BEEN UPDATED

Routing code: 3

Message issued by segment: DPPSWRST

Explanation: The status of a DDS changed or was being initialized and the DDSTATUS data set was updated.

Response: None.

DPP886 DDSTATUS RECORD HAS MISSING DDSNAMES - USING CURRENT DECLARATIONS

Routing code: 3

Message issued by segment: DPPSRSTR

Explanation: The DDS declaration for the primary CPU has at least one missing declaration from the backup CPU.

Response: None.

DPP887 DDSTATUS RECORD HAS EXTRA DDNAMES, SETTING THEM IN SINGLE MODE

Routing code: 3

Message issued by segment: DPPSRSTR

Explanation: The primary CPU had at least one DDS declaration that the backup did not have.

Response: None.

DPP888 DDS RESTART IS COMPLETE

Routing code: 3

Message issued by segment: DPPSRSTR

Explanation: The refresh of the DDS status has been completed at failover/restart time.

Response: None.

DPP889 COPY REQUEST REJECTED, RUNNING IN SINGLE MODE

Routing code: 3

Message issued by segment: DPPSCRBK

Explanation: A DDS CREATE was attempted against a data set not declared duplicate.

Response: None.

DPP890 UNABLE TO OPEN DD STATUS FOR INPUT RUNNING WITH OLD BACKUP CPU DECLARATIONS

Routing code: 3

Message issued by segment: DPPSRSTR

Explanation: At failover/restart time, the DD status data set could not be opened for input. The DDS declarations of the backup CPU were used.

Response: None.

DPP891 SYNAD READING DDSTATUS RECORD, RUNNING WITH OLD BACKUP CPU DECLARATIONS

Routing code: 3

Message issued by segment: DPPSRSTR

Explanation: At failover/restart time, a synad occurred trying to read the DDSTATUS record. The backup CPU DD2 declarations will be used.

Response: None.

DPP892 EOD READING DDSTATUS RECORD, RUNNING WITH OLD BACKUP CPU DECLARATIONS

Routing code: 3

Message issued by segment: DPPSRTCP

Explanation: At failover/restart time, the attempt to read the DDSTATUS record resulted in End of data. The DDS declarations for the backup CPU will be used.

Response: None.

DPP893 LOAD MODULE cccccccc NOT FOUND BY PLAYBACK

Routing code: message is issued as an OS/VS WTO

Message issued by segment: DPPXDPB

Explanation: A load module name was passed to playback that could not be found in the specified JOBLIB of STEPLIB DD cards.

Response: Resubmit the playback job with a valid load module name and/or STEPLIB data set.

DPP894 INVALID START OR STOP DATA PASSED TO PLAYBACK

Routing code: message is issued as an OS/VS WTO

Message issued by segment: DPPXDPB

Explanation: A start or stop date was passed to playback that could not be converted to a valid julian date.

Response: Resubmit the playback job with correct dates in the following format:

DD/MMM/YY

where
DD is the day of year
MMM is month of year (only first 3 letters of month are specified Jan-Dec)
YY is year.

DPP0895 DATA RECORD DISABLED DUE TO UNUSUAL CONDITIONS

Routing code: 1

Message issued by segment: DPPXDRC

Explanation: Data record disabled due to one of the following conditions:

- ABEND in data recording task (DPPXPRINT)
- I/O errors
- Data record data set reached end of volume.

Response: Data record may be restarted (enabled) after it has been disabled by the Special Real Time Operating System if one of the following conditions are found:

- The data set is still usable and was allocated with a DISP=OLD or NEW.
- The data set was allocated with DISP=MOD and with space remaining on the data set.

DPP896 NO DATA FOUND BY PLAYBACK WITHIN SPECIFIED TIME AND ID RANGE

Routing code: Message is issued as an OS/VS WTO

Message issued by segment: DPPXDPB

Explanation: No data was found by playback within specified time and ID range.

Response: Assure that the time and date are properly specified on the playback request and that the data set specified does contain data within that time interval.

DPP897 INCOMPLETE RECORD FOUND BY PLAYBACK

Routing code: The message is issued as an OS/VS1 WTO

Message issued by segment: DPPXDPB

Explanation: The BLKSIZE and LRECL (record length) specified on the DPBIN DD card is too small to contain the largest record on the data set.

Response: The BLKSIZE and LRECL on the DPBIN DD card must be equal to the maximum BLKSIZE and LRECL used when the data was recorded.

DPP898 DATA SET NOT OPEN FOR DDNAME cccccccc. ROUTE CODES WHICH SPECIFY THIS DDNAME OUT OF SERVICE

Routing code: The message is issued as an OS/VS1 WTO

Message issued by segment: DPPMINIT

Explanation: The specified DD name was defined in the message routing code table (RCT) during SYSGEN by the MSGRC macro, but no DD card with that name could be found in the JCL. ALL routing codes referencing the specified DD name are put out of service.

Response: A DD card with the specified DD name should be placed in the JCL.

OFFLINE UTILITY MESSAGES

DPPXDB01 DATA BASE FINAL PHASE PROCESSOR ENTERED

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: The data base final phase processor has been successfully entered through execution of a LINK supervisor call by the Offline Utility.

Response: None.

DPPXDB02 INSUFFICIENT DIRECTORY SPACE ALLOCATED

Routing code: SYSPRINT

Message issued by segment: DPPXDBAT

Explanation: The data base partitioned data set does not have enough directory blocks allocated to hold all the arrays being added to the data base. The data base remains as it was prior to this execution of the data base final phase processor. Test mode is set and a return code of 12 is returned on completion.

Response: The data base partitioned data set must be either:

- Scratched and reallocated with a larger number of directory blocks specified, or
- Copied to a new data set which has been allocated with a larger number of directory blocks allocated.

DPPXDB03 DATA BASE FINAL PHASE PROCESSOR COMPLETION CODE = XX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: The data base final phase processor has completed execution and is returning control to the Offline Utility. XX is a return code with the following meanings:

Code	Description
00	Successful completion.
04	An error, indicated by previous messages, has occurred but processing continued.
08	No arrays defined for this control card.
12	Test mode set -- An explanation exists in previous messages.
16	The data set defined by the DBINIT DD card could not be opened.

The data base is modified only if the return code is 00 or 04. For any other return code, the data base remains as it was prior to this execution of the data base final phase processor.

Response: If the return code is not 00, make the changes necessary to correct the errors indicated by messages or the return code.

DPPXDB04 INVALID OPTION RECEIVED - TEST MODE ASSUMED

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: The processing mode specified is not ADD, REPL, DEL, or TEST, so the default mode of TEST is assumed. No changes are made to the data base. A return code of 12 is returned on completion.

Response: Correct the OPTION= operand on the Offline Utility control card and rerun the job.

DPPXDB05 NO ARRAYS DEFINED - NO PROCESSING PERFORMED

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: The input to the data base final phase processor did not define any arrays; therefore, no processing could be performed. A return code of 08 is returned on completion.

Response: Correct input and rerun the job.

DPPXDB06 NO PROCESSING FOR DUP ARRAY NAME - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name or number as specified on the NAME=or NUMBER= operand of the ARRAY macro.

An attempt has been made to add the named array to the data base, but an array with the same name already exists on the data base. Processing for the named array is bypassed, and a return code of 04 is set on completion of execution of the data base final phase processor.

Response: Change the name of the array named in the message and rerun the job.

DPPXDB07 UNABLE TO OPEN DATA BASE DDNAME - DBINIT

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: The data base partitioned data set defined by the DBINIT DD card cannot be opened. No processing is performed, and a return code of 16 is returned on completion.

Response: Correct the DBINIT DD card and rerun the job.

DPPXDB08 TEST MODE SET - DUP ITEM NAME - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAT

Explanation: XXXXXXXX is an Item name as specified on the NAME= operand of the ITEM macro.

An attempt has been made to add the named item to the data base, but an item with the same name already exists on the data base. The remainder of the input is processed in TEST mode, and the data base will remain as it was prior to this execution of the data base final phase processor. A completion code of 12 is returned on completion.

Response: Change the name of the ITEM being added to the data base or delete the existing data base array that contains the item name being duplicated.

DPPXDB09 DATA SIZE GT BLKSIZE - TRUNCATION FOR ARRAY NAME - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The amount of data specified for a block in the named array is greater than the array block size defined on the ARRAY macro. The data is truncated to the array block size and processing is continued. A return code of 04 is returned on completion.

Response: Increase the array block size, or reduce the amount of data for the named array, and rerun the job.

DPPXDB10 ARRAY BLOCK SIZE REDUECED TO DATA SET SIZE FOR ARRAY - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The array block size for the named array is greater than the data base data set block size. The array block size is reduced to the data set block size, and processing is continued. A return code of 04 is returned on completion.

Response: Reduce the array block size or reallocate the data set with a larger block size and rerun the job.

DPPXDB11 ARRAY ADDED - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The named array has been added to the data base.

Response: None.

DPPXDB12 ARRAY DELETED - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The named array has been deleted from the data base.

Response: None.

DPPXDB13 ARRAY REPLACED - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The named array has been replaced on the data base.

Response: None.

DPPXDB14 ARRAY TESTED IN REPLACE MODE - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The named array was successfully processed in TEST mode as if a replace operation were being done. The data base is not modified.

Response: None.

DPPXDB15 ARRAY NOT FOUND - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. An attempt was made to replace or delete the named array, but the array did not exist on the data base.

Response: When processing in DEL mode, ensure that the array name is correct. When processing in REPL mode, ensure that the array name is correct or that a new array is being added to the data base.

DPPXDB16 BLOCK COUNT EXCEEDED FOR ARRAY - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The number of blocks of data specified on BLOCK macros for the named array is greater than the block count specified on the ARRAY macro. Excessive blocks of data will not be processed. A rerun code of 04 will be returned on completion.

Response: Increase the block count on the ARRAY macro, or reduce the number of blocks of data specified on BLOCK macros. After corrections are made, rerun the job.

DPPXDB17 DUMMY BIT SET - NO PROCESSING FOR ARRAY - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The dummy bit has been set for the named array. The array will not be processed. There will be either another data base error message for the array or an MNOTE at the time the array was assembled. A rerun code of 04 will be returned on completion.

Response: Correct the error indicated by a message or an MNOTE and rerun the job.

DPPXDB18 RC=8 FROM BLDL - PERM I/O ERROR ON ARRAY - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. A permanent I/O error indication has been returned by the BLDL SVC while trying to read the directory entry for the named array. Processing for this array is bypassed. A rerun code of 04 will be returned on completion.

Response: Determine and correct the cause of the I/O error and rerun the job.

Note: Ensure that the data base partitioned data set has been allocated as a partitioned data set and not a sequential data set.

DPPXDB19 TEST MODE ENTERED - DUPLICATE ARRAY IN INPUT - XXXXXXXX IN INPUT - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBAS

Explanation: XXXXXXXX is an array name. The input contains two arrays with the same name. Processing will continue in test mode, and a rerun code of 12 will be returned on completion.

Response: Correct the array names and rerun the job.

DDPXDB25 TEST MODE ENTERED - UNABLE TO OPEN DDNAME - XXXXXXXX
OPEN DDNAME - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBLG

Explanation: XXXXXXXX is a DD name for a data base BDAM data set. The named DD The named DD statement could not be opened for data base processing. The remainder of the input is processed in test mode. The data base is not modified by this execution. A return code of 12 will be returned on completion.

Response: Correct the DD statement or the ARRAY macro that specified the ddname and rerun the job.

DPPXDB35 RUN ABORTED - UNABLE TO OPEN ddname - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment - DPPXDBCP

Explanation: XXXXXXXX is the name of a DD statement. The named DD statement is required for the execution of the COMPRESS No processing is performed.

Response: Correct the DD statement and rerun the job.

DPPXDB36 INVALID DATA BASE DATA SET: ddname - DBINIT

Routing code: SYSPRINT

Message issued by segment - DPPXDBCP

Explanation: The data set described by the DBINIT DD statement is not a valid data base partitioned data set. No processing is performed.

Response: Correct the DD statement and rerun the job.

DPPXDB37 RC=8 FROM BLDL - PERM I/O ERROR

Routing code: SYSPRINT

Message issued by segment - DPPXDBCP

Explanation: A permanent I/O error indication was returned by the BLDL SVC while processing the DBINIT DD statement. No processing is performed.

Response: Determine and correct the cause of the I/O error and rerun the job.

Note: Ensure that the DBINIT DD statement describes a partitioned data set and not a sequential data set.

DPPXDB38 DATA BASE DOES NOT CONTAIN DIRECT ACCESS ARRAYS

Routing code: SYSPRINT

Message issued by segment: DPPXDBCP

Explanation: No compress operations can be performed, since the data base contains no direct access arrays.

Response: None.

DPPXDB39 DATA BASE COMPRESS COMPLETED FOR ddname - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DDPXDBCP

Explanation: XXXXXXXX is a DD statement name. The data base BDAM data set described by the named DD statement has been compressed, and all appropriate changes have been made to the PDS described by the DBINIT DD statement.

Response: None.

DPPXDB40 ****** THE SPECIAL REAL TIME OPERATING SYSTEM DATA BASE BDAM DATA SET COMPRESS ******

Routing code: SYSPRINT

Message issued by segment: DPPXDBCP

Explanation: This message indicates that the data base BDAM data set compress program has started execution.

Response: None.

DPPXDB41 ****** END OF DATA BASE BDAM DATA SET COMPRESS ******

Routing code: SYSPRINT

Message issued by segment: DPPXDBCP

Explanation: This message indicates that the data base BDAM data set compress program has completed execution.

Response: None.

DPPXDB42 NO DD STATEMENT INCLUDED FOR ddname - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DDPXDBCP

Explanation: XXXXXXXX is a DD statement name. The data base contains direct access arrays which are referenced by the named DD statement, but the JCL does not contain the DD statement. The data set referenced by the named DD statement is not compressed.

Response: Include the DD statement in the JCL and rerun the job.

DPPXDB50 TEST MODE ENTERED - UNABLE TO OPEN ddname - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DDPXDBDA

Explanation: XXXXXXXX is a DD name for a data base BDAM data set. The named DD statement could not be opened for data base processing. The remainder of the input is processed in test mode. The data base is not modified by this execution. A return code of 12 will be returned on completion.

Response: Correct the DD statement or the ARRAY macro which specified the DD name and rerun the job.

DPPXDB51 TEST MODE ENTERED - NO PROCESSING FOR UNBLOCKED DA ARRAY XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBDA

Explanation: XXXXXXXX is an array name. The array macro for the named array specified the operand LOCATE=DA but describes the array as unblocked. The array cannot be processed, since all DA arrays must be blocked. The remainder of the input is processed in TEST mode, and the data base is not modified by this execution. A return code of 12 is returned on completion.

Response: Correct the array macro and rerun the job.

DPPXDB52 ARRAY BLOCK SIZE REDUCED TO DATA SET BLOCK SIZE FOR ARRAY - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBDA

Explanation: XXXXXXXX is an array name. The array block size for the named array is greater than the data base data set block size. The array block size is reduced to the data set block size and processing is continued. A return code of 04 is returned on completion.

Response: Reduce the array block size or reallocate the data set with a larger block size and rerun the job.

DPPXDB53 DATA SIZE GT BLKSIZE - TRUNCATION FOR ARRAY NAME - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXDBDA

Explanation: XXXXXXXX is an array name. The amount of data specified for a block in the named array is greater than the array block size defined on the ARRAY macro. The data is truncated to the array block size and processing is continued. A return code of 04 is returned on completion.

Response: Increase the array block size or reduce the amount of data for the named array and rerun the job.

DPPXDB54 BLOCK COUNT EXCEEDED FOR ARRAY - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment - DPPXDBDA

Explanation: XXXXXXXX IS AN ARRAY NAME. The number of blocks of data specified on BLOCK macros for the named array is greater than the block count specified on the ARRAY macro. Excessive blocks of data will not be processed. A return code of 04 will be returned on completion.

Response: Increase the block count on the ARRAY macro or reduce the number of blocks of data specified on BLOCK macros. After corrections are made, rerun the job.

DPPXUT01 MISSING DDCARD - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: XXXXXXXX is a DD statement name. The named DD statement is required but is not included in the JCL. DD statements may be required because it is specified on the INPUT= operand of the control card or may be required for offline utility execution.

Response: Correct the control card or DD statement name and rerun the job.

DPPXUT02 FIRST CARD MUST BE A CONTROL CARD

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The first card read from the SYSIN data set must be a valid offline utility control card. If it is not, no processing will be done.

Response: Correct the SYSIN input and rerun the job.

DPPXUT03 PARAMETER OR CONTINUATION MARK MISSING

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The control card being processed is missing a required parameter, or a continuation mark is missing if the control card is continued on another card. Processing for this control card is bypassed. Processing will commence with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT04 EXPECTED CONTINUATION NOT RECEIVED

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The control card being processed indicated that a continuation card existed but no continuation card was received. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT05 COLUMNS 1-15 MUST BE BLANK

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: Card columns 1 through 15 must be left blank on control card continuations. on control card continuations. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT06 CONTROL CARD TEXT BEYOND COL 71

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The text of a control card must not extend past card column 71. If more space is needed, then continuation cards must be used. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT07 WRONG PARAMETER: XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: XXXXXXXX is the parameter in error. An invalid value has been specified for one of the operands on the control card. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT08 MULTIPLE KEYWORD: XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: XXXXXXXX is a keyword operand on the offline utility control card. The named keyword operand has been specified more than once on the same control card. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT09   PARAMETER IN ERROR: XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: XXXXXXXX is a parameter specified on the offline utility control card. The named parameter is invalid. Processing will continue with the next control card.

Response: Correct the parameter and rerun the job.

DPPXUT10   RIGHT PARENTHESIS MISSING - TREATED AS VALID

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: One of the parameters on the offline utility control card was started with a left parenthesis but not ended with a right parenthesis. with a right parenthesis. Processing continues as if the right parenthesis were present.

Response: Correct the control card.

DPPXUT11   WRONG KEYWORD: XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: XXXXXXXX is a keyword operand on an offline utility control card. The named keyword operand is invalid. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT12   INPUT SPECIFICATION MISSING

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The INPUT= operand is required on the DPPXUCTL control card, but it has been omitted. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT13   AREA SPECIFICATION MISSING

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The AREA= operand is required on the DPPXUCTL control card, but it has been omitted. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT15 NEW SET SPECIFICATION MISSING

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The NEW SET= operand is required on the DPPXUPDT control card, but it has been omittted. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT16 OLDSET SPECIFICATION MISSING

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The OLDSET= operand is required on the DPPXUPDT control card, but it has been omitted. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT17 NO OPERAND FOUND

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: A DPPXUPDT or DPPXUCTL control card has been encountered, but no operands were specified. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT18 INVALID OPERATION

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: An offline utility control card has been encountered, but the operation field is invalid. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT19 NO OPERATION FOUND

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: An offline utility control card has been encountered, but no operation or operands have been specified. Processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT20 SYSIN END-OF-FILE

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: An end-of-file has been encountered on the data set described by the SYSIN DD statement.

Response: None.

DPPXUT21 CONTROL CARD INVALID, SKIPPING FOR NEXT CONTROL CARD

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: An invalid offline utility control card has been encountered, and processing will continue with the next control card.

Response: Correct the control card and rerun the job.

DPPXUT22 ****** THE SPECIAL REAL TIME OPERATING SYSTEM OFFLINE UTILITY DPPXUTIL ******

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The Special Real Time Operating System offline utility program has started execution.

Response: None.

DPPXUT23 ****** END OF THE SPECIAL REAL TIME OPERATING SYSTEM OFFLINE UTILITY DPPXUTIL ******

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The Special Real Time Operating System offline utility program has completed execution.

Response: None.

DPPXUT24 END-OF-FILE ON INPUT DATA SET

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: An end-of-file has been reached on the input data set described by the INPUT= operand of the DPPXUCTL control card.

Response: None.

DPPXUT25 PARM FIELD INVALID - PARM='F,NOGEN' ASSUMED

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: The value specified in the PARM field of the execute card is invalid. The default PARM value of 'F,NOGEN' will be assumed.

Response: Correct the PARM field on the EXEC card and rerun the job.

DPPXUT26 PROCESSING ABORTED DUE TO BAD RETURN CODE FROM - XXXXXXXX

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation: XXXXXXXX is either ASSEMBLY or LOADER. A return code of 8 or greater from the assembler or from the loader will cause processing to be aborted for the current control card. Processing will continue with the next control card.

Response: Correct the errors indicated by the assembler or the loader and rerun the job.

DPPXUT99 CONTROL CARD ACCEPTED

Routing code: SYSPRINT

Message issued by segment: DPPXUTIL

Explanation:The current control card has been accepted by the offline utility program for processing.

Response: None.

APPENDIX A. THE SPECIAL REAL-TIME OPERATING SYSTEM SAMPLE PROGRAM

The Special Real Time Operating System Sample Program provides a minimal test of the functioning of the Special Real Time Operating System. It also provides a demonstration of the Special Real Time Operating System. It can be used as an example and training tool, as well as an instructional aid for application program education. The sample program consists of two programs (DPPZSAMP, DPPSAMP1).

DPPZSAMP will test and provide examples of the following Special Real Time Operating System subsystems:

Task Management (PATCH Macro)
Data Base Management (GETARRAY, GETITEM, PUTARRAY, PUTITEM,
  GETLOG and PUTLOG Macros).
TIME Management (PTIME Macro)
Realtime Message Handler (MESSAGE Macro).

DPPZSAMP will require the following array (DPPZSAMP} to be built by the Special Real Time Operating System Offline Utility:

```
#/      DPPXUCTL     AREA=DBDEF,INPUT=*,OPTION=ADD
ARRAY   NAME=DPPZSAMP,INIT=YES,REINIT=YES,
        LOCATE=VS,LOGNAME=DPPSAMP1,
        LOGDD=DBINIT2,LOGFREQ=0
  ITEM  NAME=DPPSAMP2,TYPE=C,LEN=6,INIT=ARRAY
  ITEM  NAME=DPPSAMP3,TYPE=C,LEN=9,INIT=DPPZSAMP
  ITEM  NAME=DPPSAMP4,TYPE=C,LEN=3,INIT=IS
  ITEM  NAME=DPPSAMP5,TYPE=C,LEN=5,INIT=USED
  ITEM  NAME=DPPSAMP6,TYPE=C,LEN=3,INIT=BY
  ITEM  NAME=DPPSAMP7,TYPE=C,LEN=6,INIT=SRTOS
  ITEM  NAME=DPPSAMP8,TYPE=C,LEN=7,INIT=SAMPLE
  ITEM  NAME=DPPSAMP9,TYPE=C,LEN=8,INIT=PROGRAM
  ITEM  NAME=DPPSAMPA,TYPE=C,LEN=4,INIT=FOR
  ITEM  NAME=DPPSAMPB,TYPE=C,LEN=5,INIT=TEST
  ITEM  NAME=DPPSAMPC,TYPE=C,LEN=8,INIT=PURPOSES
```

The following example is typical of the JCL required to define the sample array. Following is a description of each of the JCL statements in the example. The underlined portions of the JCL will likely have to be changed by the user to suit the requirements of his operation.

```
//BUILD          JOB     (ACCOUNTING INFORMATION)
//S1             EXEC           PGM=DPPXUTIL,PARM=H
//STEPLIB        DD      DSN=USER.PROCLIB,DISP=SHR
//SYSPRINT       DD      SYSOUT=A
//ASMPRINT       DD      SYSOUT=A
//LODPRINT       DD      DUMMY
//SYSLIB         DD      DSN=USER.MACLIB,DISP=SHR
//               DD      DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1         DD      UNIT=(SYSDA,SEP=SYSLIB),SPACE=(CYL,(2,2))
//SYSUT2 *       DD      UNIT=(SYSDA,SEP=SYSUT1),SPACE=(CYL,(2,2))
//SYSUT3 *       DD      UNIT=(SYSDA,SEP=SYSUT1),SPACE=(CYL,(2,2))
//SYSUT4         DD      UNIT=(SYSDA,SEP=SYSUT1),SPACE=(CYL,(2,2))
//                       DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//DBINIT         DD      DSN=USER.DB1,DISP=OLD
//DBINIT2        DD      DSN=USER.DB2,DISP=(MOD,PASS),DCB=(DSORG=DA)
//SYSGO          DD      UNIT=SYSDA,SPACE=(CYL,(1,1)),
                             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSIN          DD      *

                         (Input Control Statements)
```

JCL Example

```
/*
```

JOB
Is a standard OS/VS1 job card; the accounting information is dependent upon individual installation requirements.

EXEC
Is a standard OS/VS1 EXEC card; it must specify PGM=DPPXUTIL or an applicable user PROC.

PARM
The offline utility will provide the option to print or not to print statements generated by the processing of a macro. This will be accomplished by the offline utility inserting or not inserting a PRINT NOGEN statement as the first statement in the Assembler SYSIN stream. Control will be provided through the PARM keyword operand on the execute card for DPPXUTIL. This option is provided in addition to the option to select the OS/VS1 assembler or the H assembler.

The following values may be specified:

F -- Selects the OS/VS1 Assembler.

H -- Selects the H Assembler.

GEN -- Print macro generated statements.

NOGEN -- Do not print macro generated statements.

In all cases, the default values will be "F" and "NOGEN".

*Not required when "PARM=H" is specified on the execute card.

Valid combinations of the values are:

```
PARM = 'F'
PARM = 'H'
PARM = 'GEN'
PARM = 'NOGEN'
PARM = 'F,GEN'
PARM = 'F,NOGEN'
PARM = 'H,GEN'
PARM = 'H,NOGEN'
```

If an invalid value is specified for the PARM operand or if the PARM operand is omitted, the default of PARM='F,NOGEN' will be used.

STEPLIB DD
Defines the library containing the DPPXUTIL program and final phase processors and is not required if these programs reside in SYS1,LINKLIB.

SYSPRINT DD
Defines a data set in which printed output will be placed, or may specify a standard output class.

ASMPRINT DD
(Same as SYSPRINT) for printed output from the assembler.

LODPRINT DD
(Same as SYSPRINT) for printed output from the loader. It is recommended that this be a DD DUMMY to reduce printed output.

SYSLIB DD
Defines the data set(s) containing the macros used by the assembler.

SYSUT1 DD
Defines the assembler work data sets. SYSDA defines a direct-access device. This name (SYSDA), if This name (SYSDA), if used, must have been generated into the OS/VS1 system. SEP= is specified to improve assembler performance.

SYSUT2 DD
(Same as SYSUT1). Not required when "PARM=H" is specified on the execute card.

SYSUT3 DD
(Same as SYSUT2).

SYSUT4 DD
Defines a work data set for DPPXUTIL. The DCB parameters must specify RECFM=FB and a BLKSIZE that is a multiple of 80. The LRECL must be 80.

DBINIT DD
Defines the data base partitioned data set that contains a member for every array in the data base, control information for direct access resident arrays and initial data for VS resident arrays. This DD card is required if any utility control card specifies AREA=DBDEF.

DBINIT2 DD
Defines the BDAM data set which contains the initial data for DA resident arrays. This DD card is required if any utility control statement specifies AREA=DBDEF. The data set described by this DD card must be allocated prior to the execution of the The DISP= operand on this DD card must be specified as (MOD,PASS).

SYSGO DD

Defines the data set to contain the object deck output from This data set is used as input to the OS/VS1 loader.

SYSIN DD
Defines the input from which DPPXUTIL gets its control statements as possibly some source macro statements.

The sample programs (DPPZSAMP and DPPSAMP1) are not copied to the target data sets at SYSGEN time. Therefore, to execute them, the user must copy them or use a STEPLIB DD card to allocate data set A5799AHE.OBJECT.

The Special Real Time Operating System Sample Program can be executed by adding the following PATCH and WAIT input cards to the Special Real Time Operating System Subsystem Initialization Stream:

```
 P1 PATCH TASK=DPPZSAMP,EP=DPPZSAMP
    WAIT P1
```

The following example is typical of the JCL required to execute the sample problem.

```
//REAL          JOB     3DE501,'PROGRAMMER',CLASS=F
//              EXEC        PGM=DPPINIT
//STEPLIB       DD      DSN=ACS370.LM01,DISP=SHR
//DBINIT        DD      DSN=ACS370.DB1,DISP=SHR
//DBINIT2       DD      DSN=ACS370.DB2,DISP=SHR
//MSGDS         DD      DSN=ACS370.DB4,DISP=SHR
//DPPFAIL       DD      DSN=ACS370.FALRST,DISP=OLD
//SYSPRINT      DD      SYSOUT=A
//MSGOUT        DD      SYSOUT=A
//SYSUDUMP      DD      SYSOUT=A
//SYSINIT       DD      *
```

In the previous example, the JOB card is standard OS, and accounting information must be as required for the individual installation. The EXEC card must specify PGM-DPPINIT. The STEPLIB DD card points to the library(ies) containing the Special Real Time Operating System and user programs. The library name will depend upon the name given the data sets at SYSGEN time. The data sets required for the data base are pointed to by the DD cards DBINIT and DBINIT2. The online message handler requires the MSGDS and MSGOUT DD cards. The SYSPRINT DD card is required by initialization to print the input control statements. A SYSUDUMP or SYSABEND DD card is optional, depending on whether a dump is required on ABEND conditions. The SYSINIT DD card is required, and it must point to the data set containing the control statements for the online run.

DPPZSAMP will issue the following messages, if all tested subsystems are functioning properly:

```
 DPP068I   HH:MM:SS.TH DD/MMM/YY PATCH MACRO FUNCTIONING
 DPP066I   HH:MM:SS.TH DD/MMM/YY ARRAY DPPZSAMP IS USED BY
           SRTOS SAMPLE PROGRAM FOR TEST PURPOSES
 DPP068I   HH:MM:SS.TH DD/MMM/YY PUTLOG MACRO FUNCTIONING
 DPP066I   HH:MM:SS.TH DD/MMM/YY ARRAY DPPZSAMP IS USED BY SRTOS
           SAMPLE PROGRAM FOR TEST PURPOSES
 DPP068I   HH:MM:SS.TH DD/MMM/YY PUTARRAY MACRO FUNCTIONING
 DPP069I   HH:MM:SS.TH DD/MMM/YY ITEM DPPSAMP2 CONTENTS ARE ARRAY
 DPP068I   HH:MM:SS.TH DD/MMM/YY PUTITEM MACRO FUNCTIONING
 DPP068I   HH:MM:SS.TH DD/MMM/YY PTIME MACRO FUNCTIONING
```

The only function of DPPSAMP1 is to issue messages. It is used to substantiate the functioning of Task Management and Time Management. DPPSAMP1 will be PATCHed four times: once due to the PATCH Macro issued by DPPZSAMP and three times on a 1-second cycle caused by the PTIME Macro issued by DPPZSAMP. Each time DPPSAMP1 is executed, the following message is issued:

```
DPP066I   HH:MM:SS.TH DD/MMM/YY TASK - DPPSAMP1 WAS ENTERED
          AT ENTRY POINT - DPPSAMP1
```

An assembler listing of DPPZSAMP and DPPSAMP1 follows:

```
                                 EXTERNAL SYMBOL DICTIONARY                                      PAGE    1

SYMBOL   TYPE  ID  ADDR  LENGTH  LD ID                                                 ASM H V 04 09.15 11/04/75

DPPZSAMP  SD 0001 000000 0004AC




    DPPZSAMP        SAMPLE PROGRAM                                                               PAGE    2

  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                            ASM H V 04 09.15 11/04/75

                                      18 ********************************************************************** 00002000
                                      19 * MODULE NAME =DPPZSAMP                                               * 00002100
                                      20 * DESCRIPTIVE NAME = SPECIAL REAL TIME OPERATING SYSTEM SAMPLE PROGRAM* 00002200
                                      21 * FUNCTION = DPPZSAMP FUNCTION IS TO PROVIDE A MINIMAL TEST OF THE    * 00003000
                                      22 *      FUNCTIONING OF THE SPECIAL REAL TIME OPERATING SYSTEM.         * 00003100
                                      23 * NOTES = IT ALSO PROVIDES A DEMONSTRATION AND CAN BE USED AS A       * 00003200
                                      24 *      TRAINING TOOL FOR APPLICATION PROGRAM EDUCATION.               * 00004000
                                      25 * DEPENDENCIES = ARRAY'DPPZSAMP'MUST BE GENERATED BY THE USER . A     * 00004100
                                      26 *      DESCRIPTION OF THE ARRAY CAN BE FOUND IN THE SPECIAL REAL TIME * 00004200
                                      27 *      OPERATING SYSTEM DOM APPENDIX 1                                * 00005000
                                      28 * RESTRICTIONS = NONE                                                 * 00005100
                                      29 * REGISTER CONVENTIONS = ALL REGS ARE ASSIGNED AS $R WHERE REGS 0-15  * 00005200
                                      30 *      ARE  $0-$15                                                    * 00006000
                                      31 * MODULE TYPE = SAMPLE PROGRAM                                        * 00006100
                                      32 *      PROCESSOR = ASSEMBLER F                                        * 00006200
                                      33 *      MODULE SIZE = 1192 DECIMAL BYTES                               * 00007000
                                      34 *      ATTRIBUTES = REENTRANT                                         * 00007100
                                      35 * ENTRY POINT = DPPZSAMP                                              * 00007200
                                      36 * INPUT = ARRAY DPPZSAMP                                              * 00008000
                                      37 * OUTPUT = SPECIAL REAL TIME OPERATING SYSTEM MESSAGES 68 , 66 , 69   * 00008100
                                      38 * RETURN = NORMAL OS/VS RETURN. NO RETURN CODES                       * 00008200
                                      39 * EXTERNAL REFERENCES                                                 * 00008300
                                      40 *      ROUTINES = DPPSAMP1                                            * 00008400
                                      41 *      DATA AREAS = SPECIAL REAL TIME OPERATING SYSTEM DATA BASE      * 00008500
                                      42 *                  (ARRAY DPPZSAMP)                                   * 00008600
                                      43 *      CONTROL BLOCKS = XCVT                                          * 00008700
                                      44 * TABLES = NONE                                                       * 00008800
                                      45 * MACROS = BEGIN,EXIT,MESSAGE,PATCH,GETARRAY,PUTLOG,GETLOG,PUTARRAY,  * 00008900
                                      46 *           GETITEM,PUTITEM,PTIME                                     * 00008910
                                      47 ********************************************************************** 00008920
                                      48 *                                                               *     00009000
                                      49 **  THE BEGIN MACRO WILL ESTABLISH AN ENTRY POINT FOR THE       **     00010000
                                      50 **  SAMPLE PROGRAM(DPPZSAMP) , A BASE REGISTER(BASE =) AND SAVE **     00011000
                                      51 **  THE CALLING PROGRAM REGISTERS(SAVEA= AND LV=)                      00012000
                                      52 *                                                               *     00013000
                                      53          BEGIN DPPZSAMP,SAVEA=(GETMAIN,WORK),BASE=(12),LV=72          00014000
000000                                54+DPPZSAMP CSECT .                    'MAIN' CONTROL SECTION             01-BEGIN
```

```
         DPPZSAMP          SAMPLE PROGRAM                                                                     PAGE    3

  LOC  OBJECT CODE      ADDR1 ADDR2  STMT   SOURCE STATEMENT                                            ASM H V 04 09.15 11/04/75
                                      56+*                                GOES THRU REGISTER EQUATE ONLY ONCE
                            00000     57+$0        EQU   0 ?              **                                         02-EQUAT
                            00001     58+$1        EQU   1 ?              **                                         02-EQUAT
                            00002     59+$2        EQU   2 ?              **                                         02-EQUAT
                            00003     60+$3        EQU   3 ?              **                                         02-EQUAT
                            00004     61+$4        EQU   4 ?              **                                         02-EQUAT
                            00005     62+$5        EQU   5 ?              **                                         02-EQUAT
                            00006     63+$6        EQU   6 ?              **IF THESE SUBSTITUTES ARE USED AS         02-EQUAT
                            00007     64+$7        EQU   7 ?              **REGISTFR NUMBERS THE CROSS-REFERENCE     02-EQUAT
                            00008     65+$8        EQU   8 ?              **TABLE WILL PROVIDE A LIST OF WHERE       02-EQUAT
                            00009     66+$9        EQU   9 ?              **EACH REGISTER WAS USED                   02-EQUAT
                            0000A     67+$10       EQU   10 ?             **                                         02-EQUAT
                            0000B     68+$11       EQU   11 ?             **                                         02-EQUAT
                            0000C     69+$12       EQU   12 ?             **                                         02-EQUAT
                            0000D     70+$13       EQU   13 ?             **                                         02-EQUAT
                            0000E     71+$14       EQU   14 ?             **                                         02-EQUAT
                            0000F     72+$15       EQU   15 ?             **                                         02-EQUAT
                            00000     73+FPR0      EQU   0                                                           02-EQUAT
                            00002     74+FPR2      EQU   2                                                           02-EQUAT
                            00004     75+FPR4      EQU   4                                                           02-EQUAT
                            00006     76+FPR6      EQU   6                                                           02-EQUAT


000000                                78+          DS    0D .             FOR BOUNDARY ALIGNMENT                     01-BEGIN
                            00000     79+          USING *,15 .           TEMPORARY BASE DECLARATION                 01-BEGIN
000000 47F0 F00E            0000E     80+          B     14(0,15)                             BRANCH AROUND ID       02-SAVE
000004 08                             81+          DC    AL1(8)                               LENGTH OF IDENTIFIER   02-SAVE
000005 C4D7D7E9E2C1D4D7               82+          DC    CL8'DPPZSAMP'                        IDENTIFIER             02-SAVE
00000D 00
00000E 90EC D00C            0000C     83+          STM   14,12,12(13)                         SAVE REGISTERS         02-SAVE
000012 5800 F034            00034     84+          L     0,TKG0001G        . LOAD SP AND LV PARAMETERS               01-BEGIN
                                      85+*         GETMAIN R,LV=(0)
000016 4510 F01A            0001A     86+          BAL   1,*+4                                INDICATE GETMAIN       02-GETMA
00001A 0A0A                           87+          SVC   10                                   ISSUE GETMAIN SVC      02-GETMA
00001C 50D1 0004            00004     88+          ST    13,4(1) .        SAVF CALLER'S SAVE AREA POINTER            01-BEGIN
000020 501D 0008            00008     89+          ST    1,8(13) .        FOR DOWNWARD SAVE AREA TRACE               01-BEGIN
000024 18D1                           90+          LR    13,1 .           ESTABLISH OWN SAVE AREA POINTER            01-BEGIN
000026 581D 0004            00004     91+          L     1,4(13) .              RESTORE 15,0,1                       01-BEGIN
00002A 98E1 100C            0000C     92+          LM    14,1,12(1)             RESTORE GET REGS                     01-BEGIN

000000                                94+WORK       DSECT .            BEGIN GETMAINED AREA                       01-BEGIN
000000                                95+          DS    9D .             OWN SAVE AREA                              01-BEGIN
00002E                                96+DPPZSAMP  CSECT                                                             01-BEGIN
                            00000     97+          USING WORK,13                                                     01-BEGIN

00002E 0700                           99+          CNOP  0,4                                                         01-BEGIN
000030 45C0 F038            00038    100+          BAL   12,*+8      .       ESTABLISH INITIAL 'MAIN' CSECT BASE     01-BEGIN
000034                               101+TKG0001M  DS    0F .             BASE REFERENCE                             01-BEGIN
000034 00000048                      102+TKG0001G      DC    AL1(0),AL3(72)     . SUBPOOL, LENGTH                    01-BEGIN
                                     103+          DROP  15                                                          01-BEGIN
                            00034    104+          USING TKG0001M,12                                                 01-BEGIN
                                     105 *                                                                 *         00015000
                                     106 **  UPON ENTRY       PASS   PARAMETERS TO DPPZSAMP AS FOLLOWS : **          00016000
                                     107 **   **************      *****************                     **          00017000
                                     108 **   *RESIGISTER 1*----> *      XCVT      *                     **          00018000
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM H V 04 09.15 11/04/75
                                   109 **   **************      *RESOURCE TABLE  *                     **     00019000
                                   110 **                       *PATCH PARAMETERS*                     **     00020000
                                   111 **                       ******************                     **     00021000
                                   112 *                                                                *     00022000
000038 5821 0000           00000   113          L  $2,0($1)               PLACE       XCVT ADDRESS IN REG 2   00023000
                                   114 *                                                                *     00024000
                                   115 ** THE FOLLOWING PATCH MACRO WILL CREATE AN INDEPENDANT TASK     *     00025000
                                   116 ** NAMED(TASK=) DPPSAMP1 . THE TASK ENTRY POINT(EP=) IS DPPSAMP1**     00026000
                                   117 ** NOTE: THE DCVTR OR DCVTLOC OPERAND SHOULD BE USED ON ONLINE  **     00027000
                                   118 **       MACROS TO INCREASE THEIR OPERATION EFFICIENCY . DCVTR  **     00028000
                                   119 ** AND DCVTLOC POINTS TO THE       XCVT.                        **     00029000
                                   120 *                                                                *     00030000
                                   121          PATCH TASK=DPPSAMP1,EP=DPPSAMP1,DCVTR=($2)                    00031000
00003C 4110 C010           00044   122+         LA    1,IHB0005           SET UP PARAM LIST ADDRESS           01-PATCH
000040 47F0 C03C           00070   123+         B     IHB0005A            BRANCH AROUND LIST                  01-PATCH
000044                             124+IHB0005  DS    0F                                                      01-PATCH
000044 C4D7D7E2C1D4D7F1            125+         DC    CL8'DPPSAMP1'       TASK NAME                           01-PATCH
00004C C4D7D7E2C1D4D7F1            126+         DC    CL8'DPPSAMP1'       ENTRY POINT NAME                    01-PATCH
000054 4040404040404040            127+         DC    CL8' '              PRTY REFERENCE NAME                 01-PATCH
00005C 00                          128+         DC    AL1(0)              FLAG BYTE                           01-PATCH
00005D 01                          129+         DC    AL1(1)              QUEUE LENGTH                        01-PATCH
00005E 0000                        130+         DC    H'0'                PRTY RELATIVE VALUE                 01-PATCH
000060 00000000                    131+         DC    A(0)                ECB ADDRESS                         01-PATCH
000064 0000000000000000            132+         DC    2F'0'               FREE LENGTH,FREE ADDRESS            01-PATCH
00006C 00000000                    133+         DC    A(0)                TCBX                                01-PATCH
                           00070   134+IHB0005A EQU   *                                                       01-PATCH
000070 41F0 C044           00078   135+         LA    15,IHP0005          SET UP LIST ADDRESS                 01-PATCH
000074                             136+         CNOP  0,4                                                     01-PATCH
000074 4500 C048           0007C   137+         BAL   0,IHP0005A          SET UP REG 0 WITH PARAM LIST        01-PATCH
                           00078   138+IHP0005  EQU   *                                                       01-PATCH
000078 0004                        139+         DC    AL2(IHP0005A-IHP0005)  LENGTH OF PARAMS                 01-PATCH
00007A 0000                        140+         DC    AL2(0)              ID                                  01-PATCH
                           0007C   141+IHP0005A  EQU *                                                        01-PATCH
00007C 18F2                        142+         LR    15,($2)             CVT ADDRESS                         02-DPPSV
00007E BFF8 C051           00085   143+         ICM   15,8,*+7            ID IN HIGH ORDER BYTE OF REG        02-DPPSV
000082 4700 0004           00004   144+         NOP   4                   CONSTANT FOR ID                     02-DPPSV
000086 440F 0004           00004   145+         EX    0,4(15)             EXECUTE SVC FROM CVT                02-DPPSV
                                   146 ** IF RETURN CODE FROM PATCH SVC IS ZERO THEN                       **     00032000
                                   147          IF F,($15),IS,ZERO,THEN                                       00033000
00008A 12FF                        148+         LTR   $15,$15                                                 01-IF
00008C 4770 C0A0           000D4   149+         BC    7,IF10007                                               01-IF
                                   150 ** OUTPUT  MESSAGE 68            ,WHICH CONTAINS MSG1(VAR=),TO             00034000
                                   151 ** THE SYSTEM CONSOLE(ROUTE=1).                                     **     00035000
                                   152            MESSAGE 68,VAR=(MSG1),DCVTR=($2),ROUTE=1                    00036000
000090                             153+         CNOP  0,4                                                     01-MESSA
000090 4510 C070           000A4   154+         BAL   1,MSG0008                                               01-MESSA
000094 01                          155+         DC    AL1(0+1)            VARIABLE COUNT                      01-MESSA
000095 01                          156+         DC    AL1(1)              ROUTING CODE COUNT                  01-MESSA
000096 0000                        157+         DC    AL2(0)               MESSAGE NUMBER                     01-MESSA
000098 00                          158+         DC    X'00'               ACTION CODE                         01-MESSA
000099 000000                      159+         DC    AL3(0)              USER  RETURN AREA                   01-MESSA
00009C 0000                        160+         DC    AL2(0)   ROUTING CODE                                   01-MESSA
00009E 00000000                    161+         DC    1AL4(0)            MESSAGE VARIABLE                     01-MESSA
0000A4                             162+MSG0008    DS  0F                                                      01-MESSA
0000A4 4100 0044           00044   163+         LA    0,68       LOAD MSG # INTO REGISTER 0                   01-MESSA
```

```
           DPPZSAMP          SAMPLE PROGRAM                                                                     PAGE    5

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT                                          ASM H V 04 09.15 11/04/75

0000A8 4001 0002            00002  164+             STH   0,2(1)  MOVE MSG # TO PARAMETER LIST                  01-MESSA
0000AC 4100 0001            00001  165+             LA    0,1  LOAD ROUTING CODE INTO REGISTER 0                01-MESSA
0000B0 4001 0008            00008  166+             STH   0,8(1)   STORE ROUTING CODE INTO PARAMETER LIST       01-MESSA
0000B4 9680 1008      00008        167+             OI    8(1),X'80'         SET HIGH BIT OF LAST ROUTINE CODE  01-MESSA
0000B8 18FF                        168+             SR    15,15              ZERO REG 15 FOR IC                 01-MESSA
0000BA 43F1 0001            00001  169+             IC    15,1(1)            # OF ROUTE CODES IN PARAMETER LIST 01-MESSA
0000BE 89F0 0001            00001  170+             SLL   15,1               LENGTH OF ROUTE CODE IN PARAM LIST 01-MESSA
0000C2 4100 C450            00484  171+             LA    0,MSG1             VARIABLE ADDR                      01-MESSA
0000C6 5001 F008            00008  172+             ST    0,8(1,15)          STORE INTO MESSAGE LIST            01-MESSA
0000CA 58F2 0020            00020  173+             L     15,32(($2))       ADDRESS OF       CVT                02-DPPSU
0000CE 58FF 0090            00090  174+             L     15,116+28(15)      MESSAGE  SUPPORT ROUTINE           02-DPPSU
0000D2 05EF                        175+             BALR  14,15             CALL SUPPORT ROUTINE                02-DPPSU
                                   176              ENDIF                                                       00037000
0000D4                             177+IF10007      DS    0H                                                    01-ENDIF
                                   178 *                                                                *       00038000
                                   179 ** THE FOLLOWING GETARRAY MACRO WILL RETRIEVE THE ADDRESS        **      00039000
                                   180 ** (TYPE=ADDR) OF ARRAY (NAME=) DPPZSAMP AND PLACE THE ADDRESS   **      00040000
                                   181 ** IN LOCATION'ARRAY'(DATA=) .                                  **      00041000
                                   182 *                                                                *       00042000
                                   183              GETARRAY NAME=DPPZSAMP,DATA=ARRAY,TYPE=ADDR,DCVTR=($2)      00043000
0000D4                             184+             CNOP  0,4                                                   01-GETAR
0000D4 4510 C0AC            000E0  185+             BAL   1,GA0011                                              01-GETAR
0000D8 C4D7D7E9E2C1D4D7            186+G0011        DC    CL8'DPPZSAMP'                                         01-GETAR
0000E0                             187+GA0011       CNOP  0,4                                                   01-GETAR
0000E0 4100 C3FC            00430  188+             LA    0,ARRAY            ADDRESS OF DATA                    01-GETAR
0000E4 58F2 0020            00020  189+             L     15,32(($2))       ADDRESS OF       CVT                02-DPPSU
0000E8 58FF 0078            00078  190+             L     15,116+4(15)       GETARRAY SUPPORT ROUTINE           02-DPPSU
0000EC 45E0 C0BE            000F2  191+             BAL   14,*+6                                                02-DPPSU
0000F0 0200                        192+             DC    AL1(2),AL1(0)                                         02-DPPSU
0000F2 BFF8 E000            00000  193+             ICM   15,8,0(14)        INSERT THE MACRO ID                 02-DPPSU
0000F6 05EF                        194+             BALR  14,15             CALL SUPPORT ROUTINE                02-DPPSU
                                   195 ** IF THE ARRAY ADDRESS WAS RETRIEVED FROM THE DATA BASE THEN   **      00044000
                                   196              IF F,($15),IS,ZERO,THEN                                     00045000
0000F8 12FF                        197+             LTR   $15,$15                                               01-IF
0000FA 4770 C228            0025C  198+             BC    7,IF10013                                             01-IF
0000FE 5830 C3FC            00430  199                L $3,ARRAY             PLACE ARRAY ADDRESS IN REG 3       00046000
                                   200 ** OUTPUT  MESSAGE  66         ,WHICH CONTAINS ARRAY(VAR=)      **      00047000
                                   201 ** DPPZSAMP,TO THE SYSTEM CONSOLE(ROUTE=1)                      **      00048000
                                   202                 MESSAGE 66,VAR=(($3)),DCVTR=($2)                         00049000
000102 0700                        203+             CNOP  0,4                                                   01-MESSA
000104 4510 C0E0            00114  204+             BAL   1,MSG0014                                             01-MESSA
000108 01                          205+             DC    AL1(0+1)           VARIABLE COUNT                     01-MESSA
000109 00                          206+             DC    AL1(0)             ROUTING CODE COUNT                 01-MESSA
00010A 0000                        207+             DC    AL2(0)              MESSAGE NUMBER                    01-MESSA
00010C 00                          208+             DC    X'00'              ACTION CODE                        01-MESSA
00010D 000000                      209+             DC    AL3(0)             USER  RETURN AREA                  01-MESSA
000110 00000000                    210+             DC    1AL4(0)           MESSAGE VARIABLE                    01-MESSA
000114                             211+MSG0014       DS   0F                                                    01-MESSA
000114 4100 0042            00042  212+             LA    0,66      LOAD MSG # INTO REGISTER 0                  01-MESSA
000118 4001 0002            00002  213+             STH   0,2(1)  MOVE MSG # TO PARAMETER LIST                  01-MESSA
00011C 1BFF                        214+             SR    15,15              ZERO REG 15 FOR IC                 01-MESSA
00011E 43F1 0001            00001  215+             IC    15,1(1)            # OF ROUTE CODES IN PARAMETER LIST 01-MESSA
000122 89F0 0001            00001  216+             SLL   15,1               LENGTH OF ROUTE CODE IN PARAM LIST 01-MESSA
000126 5031 F008            00008  217+             ST    $3,8(1,15)         STORE VARIABLE INTO PARAMETER LIST 01-MESSA
00012A 58F2 0020            00020  218+             L     15,32(($2))       ADDRESS OF       CVT                02-DPPSU
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                               ASM H V 04 09.15 11/04/75

00012E 58FF 0090            00090   219+         L     15,116+28(15)       MESSAGE  SUPPORT ROUTINE          02-DPPSU
000132 05EF                         220+         BALR  14,15              CALL SUPPORT ROUTINE                02-DPPSU
                                    221 *                                                            *     00050000
                                    222 ** THE FOLLOWING PUTLOG MACRO WILL LOG OUT ARRAY(NAME=) DPPZSAMP**     00051000
                                    223 *                                                            *     00052000
                                    224              PUTLOG NAME=DPPZSAMP,DCVTR=($2)                       00053000
000134 4510 C10C            00140   225+         BAL   1,*+12              A(ARRAY NAME)                       01-PUTLO
000138 C4D7D7E9E2C1D4D7             226+         DC    CL8'DPPZSAMP'       ARRAY NAME                          01-PUTLO
000140 58F2 0020            00020   227+         L     15,32(($2))        ADDRESS OF       CVT                 02-DPPSU
000144 58FF 00B8            000B8   228+         L     15,116+68(15)                                           02-DPPSU
000148 45E0 C11A            0014E   229+         BAL   14,*+6                                                  02-DPPSU
00014C 0100                         230+         DC    AL1(1),AL1(0)                                           02-DPPSU
00014E BFF8 E000            00000   231+         ICM   15,8,0(14)         INSERT THE MACRO ID                  02-DPPSU
000152 05EF                         232+         BALR  14,15              CALL SUPPORT ROUTINE                 02-DPPSU
                                    233 ** IF ARRAY DPPZSAMP WAS LOGGED OUT THEN                       **     00054000
                                    234              IF F,($15),IS,ZERO,THEN                                00055000
000154 12FF                         235+         LTR   $15,$15                                                 01-IF
000156 4770 C228            0025C   236+         BC    7,IF20018                                               01-IF
                                    237 ** OUTPUT  MESSAGE  68                WHICH CONTAINS MSG2(VAR=) TO  **     00056000
                                    238 ** THE SYSTEM CONSOLE(ROUTE=1).                                       00057000
                                    239                MESSAGE 68,VAR=(MSG2),DCVTR=($2)                     00058000
00015A 0700                         240+         CNOP  0,4                                                     01-MESSA
00015C 4510 C138            0016C   241+         BAL   1,MSG0019                                               01-MESSA
000160 01                           242+         DC    AL1(0+1)              VARIABLE COUNT                    01-MESSA
000161 00                           243+         DC    AL1(0)                ROUTING CODE COUNT                01-MESSA
000162 0000                         244+         DC    AL2(0)                 MESSAGE NUMBER                   01-MESSA
000164 00                           245+         DC    X'00'                 ACTION CODE                       01-MESSA
000165 000000                       246+         DC    AL3(0)                USER  RETURN AREA                 01-MESSA
000168 00000000                     247+         DC    1AL4(0)             MESSAGE VARIABLE                    01-MESSA
00016C                              248+MSG0019   DS   0F                                                      01-MESSA
00016C 4100 0044            00044   249+         LA    0,68       LOAD MSG # INTO REGISTER 0                   01-MESSA
000170 4001 0002            00002   250+         STH   0,2(1)  MOVE MSG # TO PARAMETER LIST                    01-MESSA
000174 1BFF                         251+         SR    15,15                 ZERO REG 15 FOR IC                01-MESSA
000176 43F1 0001            00001   252+         IC    15,1(1)               # OF ROUTE CODES IN PARAMETER LIST   01-MESSA
00017A 89F0 0001            00001   253+         SLL   15,1                  LENGTH OF ROUTE CODE IN PARAM LIST   01-MESSA
00017E 4100 C458            0048C   254+         LA    0,MSG2                VARIABLE ADDR                     01-MESSA
000182 5001 F008            00008   255+         ST    0,8(1,15)             STORE INTO MESSAGE LIST           01-MESSA
000186 58F2 0020            00020   256+         L     15,32(($2))          ADDRESS OF       CVT               02-DPPSU
00018A 58FF 0090            00090   257+         L     15,116+28(15)         MESSAGE  SUPPORT ROUTINE          02-DPPSU
00018E 05EF                         258+         BALR  14,15                CALL SUPPORT ROUTINE               02-DPPSU
                                    259 *                                                            *     00059000
                                    260 ** THE FOLLOWING GETLOG MACRO WILL LOG IN ARRAY(NAME=) DPPZSAMP **     00060000
                                    261 ** AND PLACE(AREA=) THE ARRAY AT LOCATION LOGCOPY .                *     00061000
                                    262 *                                                            *     00062000
                                    263                GETLOG NAME=DPPZSAMP,AREA=LOGCOPY,DCVTR=($2)          00063000
000190                              264+         CNOP  0,4                                                     01-GETLO
000190 4510 C178            001AC   265+         BAL   1,*+28                BRANCH ARROUND PARMS              01-GETLO
000194 000001A4                     266+         DC    AL1(0),AL3(*+15)            ARRAY IDENTIFIER            01-GETLO
000198 00000438                     267+         DC    A(LOGCOPY)             OUTPUT AREA                      01-GETLO
00019C 00000000                     268+         DC    A(0)                   RELATIVE COPY                    01-GETLO
0001A0 00000000                     269+         DC    A(0)                   LOG COPY REFERENCE               01-GETLO
0001A4 C4D7D7E9E2C1D4D7             270+         DC    CL8'DPPZSAMP'                                           01-GETLO
0001AC 58F2 0020            00020   271+         L     15,32(($2))          ADDRESS OF       CVT               02-DPPSU
0001B0 58FF 00B4            000B4   272+         L     15,116+64(15)                                           02-DPPSU
0001B4 05EF                         273+         BALR  14,15                CALL SUPPORT ROUTINE               02-DPPSU
```

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                      ASM H V 04 09.15 11/04/75

                                  274 ** IF ARRAY DPPZSAMP WAS LOGGED IN THEN                                     00064000
                                  275                   IF F,($15),IS,ZERO,THEN                                   00065000
0001B6 12FF                       276+          LTR   $15,$15                                                     01-IF
0001B8 4770 C228           0025C  277+          BC    7,IF30023                                                   01-IF
                                  278 ** OUTPUT  MESSAGE 66          ,WHICH CONTAINS THE LOGGED IN      **        00066000
                                  279 ** ARRAY(VAR=) DPPZSAMP, TO THE SYSTEM CONSOLE(ROUTE=1)                     00067000
0001BC 4140 C41C           00450  280                   LA  $4,LOGCOPY+24 LOGGED ARRAY ADDRESS(1ST 24BYTES OF A   00068000
                                  281 **                                    LOGGED ARRAY CONTAINS A HEADER        00068100
                                  282                   MESSAGE 66,VAR=(($4)),DCVTR=($2),ROUTE=1                  00068200
0001C0                            283+          CNOP  0,4                                                         01-MESSA
0001C0 4510 C1A0           001D4  284+          BAL   1,MSG0024                                                   01-MESSA
0001C4 01                         285+          DC    AL1(0+1)             VARIABLE COUNT                         01-MESSA
0001C5 01                         286+          DC    AL1(1)               ROUTING CODE COUNT                     01-MESSA
0001C6 0000                       287+          DC    AL2(0)                MESSAGE NUMBER                        01-MESSA
0001C8 00                         288+          DC    X'00'                ACTION CODE                            01-MESSA
0001C9 000000                     289+          DC    AL3(0)               USER  RETURN AREA                      01-MESSA
0001CC 0000                       290+          DC    AL2(0)   ROUTING CODE                                       01-MESSA
0001CE 00000000                   291+          DC    1AL4(0)             MESSAGE VARIABLE                        01-MESSA
0001D4                            292+MSG0024    DS   0F                                                          01-MESSA
0001D4 4100 0042           00042  293+          LA    0,66      LOAD MSG # INTO REGISTER 0                        01-MESSA
0001D8 4001 0002           00002  294+          STH   0,2(1)  MOVE MSG # TO PARAMETER LIST                        01-MESSA
0001DC 4100 0001           00001  295+          LA    0,1  LOAD ROUTING CODE INTO REGISTER 0                      01-MESSA
0001E0 4001 0008           00008  296+          STH   0,8(1)   STORE ROUTING CODE INTO PARAMETER LIST             01-MESSA
0001E4 9680 1008     00008        297+          OI    8(1),X'80'           SET HIGH BIT OF LAST ROUTINE CODE      01-MESSA
0001E8 18FF                       298+          SR    15,15                ZERO REG 15 FOR IC                     01-MESSA
0001EA 43F1 0001           00001  299+          IC    15,1(1)              # OF ROUTE CODES IN PARAMETER LIST     01-MESSA
0001EE 89F0 0001           00001  300+          SLL   15,1                 LENGTH OF ROUTE CODE IN PARAM LIST     01-MESSA
0001F2 5041 F008           00008  301+          ST    $4,8(1,15)           STORE VARIABLE INTO PARAMETER LIST     01-MESSA
0001F6 58F2 0020           00020  302+          L     15,32(($2))         ADDRESS OF       CVT                      02-DPPSU
0001FA 58FF 0090           00090  303+          L     15,116+28(15)        MESSAGE  SUPPORT ROUTINE               02-DPPSU
0001FE 05EF                       304+          BALR  14,15               CALL SUPPORT ROUTINE                      02-DPPSU
                                  305 *                                                                 *         00069000
                                  306 ** THE FOLLOWING PUTARRAY MACRO WILL      PLACE(DATA=) THE        **        00070000
                                  307 ** LOGGED IN ARRAY(NAME=) DPPZSAMP IN THE DATA BASE .                       00071000
                                  308 *                                                                 *         00072000
                                  309                   PUTARRAY NAME=DPPZSAMP,DATA=($4),DCVTR=($2)               00073000
000200                            310+          CNOP  0,4                                                         01-PUTAR
000200 4510 C1D8           0020C  311+          BAL   1,PA0026                                                    01-PUTAR
000204 C4D7D7E9E2C1D4D7           312+A0026     DC    CL8'DPPZSAMP'       ARRAY NAME                              01-PUTAR
00020C                            313+PA0026    CNOP  0,4                                                         01-PUTAR
00020C 1804                       314+          LR    0,$4                ADDRESS OF DATA                         01-PUTAR
00020E 58F2 0020           00020  315+          L     15,32(($2))         ADDRESS OF       CVT                    02-DPPSU
000212 58FF 007C           0007C  316+          L     15,116+8(15)         PUTARRAY SUPPORT ROUTINE               02-DPPSU
000216 45E0 C1E8           0021C  317+          BAL   14,*+6                                                      02-DPPSU
00021A 7000                       318+          DC    AL1(112),AL1(0)                                             02-DPPSU
00021C BFF8 E000           00000  319+          ICM   15,8,0(14)          INSERT THE MACRO ID                     02-DPPSU
000220 05EF                       320+          BALR  14,15               CALL SUPPORT ROUTINE                    02-DPPSU
                                  321 ** IF THE LOGGED ARRAY WAS PLACED IN THE DATA BASE THEN                     00074000
                                  322                   IF F,($15),IS,ZERO,THEN                                   00075000
000222 12FF                       323+          LTR   $15,$15                                                     01-IF
000224 4770 C228           0025C  324+          BC    7,IF40028                                                   01-IF
                                  325 ** OUTPUT MESSAGE  68,WHICH  CONTAINS MSG3(VAR=), TO THE          **        00076000
                                  326 ** SYSTEM CONSOLE(ROUTE=1)                                        **        00077000
                                  327                     MESSAGE 68,VAR=(MSG3),DCVTR=($2)                        00078000
000228                            328+          CNOP  0,4                                                         01-MESSA
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                             ASM H V 04 09.15 11/04/75

000228 4510 C204            00238   329+          BAL   1,MSG0029                                                  01-MESSA
00022C 01                           330+          DC    AL1(0+1)             VARIABLE COUNT                        01-MESSA
00022D 00                           331+          DC    AL1(0)               ROUTING CODE COUNT                    01-MESSA
00022E 0000                         332+          DC    AL2(0)                MESSAGE NUMBER                       01-MESSA
000230 00                           333+          DC    X'00'                ACTION CODE                           01-MESSA
000231 000000                       334+          DC    AL3(0)               USER  RETURN AREA                     01-MESSA
000234 00000000                     335+          DC    1AL4(0)             MESSAGE VARIABLE                       01-MESSA
000238                              336+MSG0029     DS   0F                                                        01-MESSA
000238 4100 0044            00044   337+          LA    0,68        LOAD MSG # INTO REGISTER 0                     01-MESSA
00023C 4001 0002            00002   338+          STH   0,2(1)  MOVE MSG # TO PARAMETER LIST                       01-MESSA
000240 1BFF                         339+          SR    15,15                ZERO REG 15 FOR IC                    01-MESSA
000242 43F1 0001            00001   340+          IC    15,1(1)              # OF ROUTE CODES IN PARAMETER LIST    01-MESSA
000246 89F0 0001            00001   341+          SLL   15,1                 LENGTH OF ROUTE CODE IN PARAM LIST    01-MESSA
00024A 4100 C460            00494   342+          LA    0,MSG3               VARIABLE ADDR                         01-MESSA
00024E 5001 F008            00008   343+          ST    0,8(1,15)            STORE INTO MESSAGE LIST               01-MESSA
000252 58F2 0020            00020   344+          L     15,32(($2))         ADDRESS OF       CVT                   02-DPPSU
000256 58FF 0090            00090   345+          L     15,116+28(15)        MESSAGE  SUPPORT ROUTINE              02-DPPSU
00025A 05EF                         346+          BALR  14,15               CALL SUPPORT ROUTINE                   02-DPPSU
                                    347                 ENDIF                                                      00079000
00025C                              348+IF40028  DS     0H                                                         01-ENDIF
                                    349               ENDIF                                                        00080000
00025C                              350+IF30023  DS     0H                                                         01-ENDIF
                                    351             ENDIF                                                          00081000
00025C                              352+IF20018  DS     0H                                                         01-ENDIF
                                    353           ENDIF                                                            00082000
00025C                              354+IF10013  DS     0H                                                         01-ENDIF
                                    355 *                                                                  *       00083000
                                    356 ** THE FOLLOWING GETITEM MACRO WILL RETRIEVE THE CONTENTS          *       00084000
                                    357 ** (TYPE=DATA) OF ITEM(NAME=) DPPSAMP2 AND PLACE(DATA=) THE DATA**         00085000
                                    358 ** AT LOCATION'ITEM'.                                             **       00086000
                                    359 *                                                                  *       00087000
                                    360           GETITEM NAME=DPPSAMP2,DATA=ITEM,TYPE=DATA,DCVTR=($2)             00088000
00025C                              361+          CNOP  0,4                                                        01-GETIT
00025C 4510 C238            0026C   362+          BAL   1,GI0035                                                   01-GETIT
000260 C4D7D7E2C1D4D7F2             363+I0035     DC    CL8'DPPSAMP2'       ITEM NAME                              01-GETIT
000268 0000047E                     364+          DC    AL1(0),AL3(ITEM)                                           01-GETIT
00026C                              365+GI0035    CNOP  0,4                                                        01-GETIT
00026C 5800 C234            00268   366+          L     0,I0035+8           ADDRESS OF DATA                        01-GETIT
000270 58F2 0020            00020   367+          L     15,32(($2))         ADDRESS OF       CVT                   02-DPPSU
000274 58FF 0080            00080   368+          L     15,116+12(15)        GETITEM  SUPPORT ROUTINE              02-DPPSU
000278 45E0 C24A            0027E   369+          BAL   14,*+6                                                     02-DPPSU
00027C 7800                         370+          DC    AL1(120),AL1(0)                                            02-DPPSU
00027E BFF8 E000            00000   371+          ICM   15,8,0(14)          INSERT THE MACRO ID                    02-DPPSU
000282 05EF                         372+          BALR  14,15               CALL SUPPORT ROUTINE                   02-DPPSU
                                    373 ** IF ITEM DPPSAMP2 WAS RETRIEVED THEN                            **       00089000
                                    374           IF F,($15),IS,ZERO,THEN                                          00090000
000284 12FF                         375+          LTR   $15,$15                                                    01-IF
000286 4770 C310            00344   376+          BC    7,IF10037                                                  01-IF
                                    377 **OUTPUT MESSAGE 69,WHICH CONTAINS (VAR=) THE ITEM, TO THE        **       00091000
                                    378 ** SYSTEM CONSOLE(ROUTE=1)                                        **       00092000
                                    379             MESSAGE 69,VAR=(ITEM),DCVTR=($2),ROUTE=1                       00093000
00028A 0700                         380+          CNOP  0,4                                                        01-MESSA
00028C 4510 C26C            002A0   381+          BAL   1,MSG0038                                                  01-MESSA
000290 01                           382+          DC    AL1(0+1)             VARIABLE COUNT                        01-MESSA
000291 01                           383+          DC    AL1(1)               ROUTING CODE COUNT                    01-MESSA
```

```
          DPPZSAMP         SAMPLE PROGRAM                                                                              PAGE    9

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                                   ASM H V 04 09.15 11/04/75

000292 0000                        384+          DC     AL2(0)                MESSAGE NUMBER                          01-MESSA
000294 00                          385+          DC     X'00'                ACTION CODE                              01-MESSA
000295 000000                      386+          DC     AL3(0)               USER  RETURN AREA                        01-MESSA
000298 0000                        387+          DC     AL2(0)   ROUTING CODE                                         01-MESSA
00029A 00000000                    388+          DC     1AL4(0)             MESSAGE VARIABLE                          01-MESSA
0002A0                             389+MSG0038     DS   OF                                                            01-MESSA
0002A0 4100 0045             00045 390+          LA     0,69       LOAD MSG # INTO REGISTER 0                         01-MESSA
0002A4 4001 0002             00002 391+          STH    0,2(1)  MOVE MSG # TO PARAMETER LIST                          01-MESSA
0002A8 4100 0001             00001 392+          LA     0,1  LOAD ROUTING CODE INTO REGISTER 0                        01-MESSA
0002AC 4001 0008             00008 393+          STH    0,8(1)   STORE ROUTING CODE INTO PARAMETER LIST               01-MESSA
0002B0 9680 1008       00008       394+          OI     8(1),X'80'           SET HIGH BIT OF LAST ROUTINE CODE        01-MESSA
0002B4 1BFF                        395+          SR     15,15                ZERO REG 15 FOR IC                       01-MESSA
0002B6 43F1 0001             00001 396+          IC     15,1(1)              # OF ROUTE CODES IN PARAMETER LIST       01-MESSA
0002BA 89F0 0001             00001 397+          SLL    15,1                 LENGTH OF ROUTE CODE IN PARAM LIST       01-MESSA
0002BF 4100 C44A             0047E 398+          LA     0,ITEM               VARIABLE ADDR                            01-MESSA
0002C2 5001 F008             00008 399+          ST     0,8(1,15)            STORE INTO MESSAGE LIST                  01-MESSA
0002C6 58F2 0020             00020 400+          L      15,32(($2))         ADDRESS OF        CVT                     02-DPPSU
0002CA 58FF 0090             00090 401+          L      15,116+28(15)        MESSAGE  SUPPORT ROUTINE                 02-DPPSU
0002CE 05EF                        402+          BALR  14,15                CALL SUPPORT ROUTINE                      02-DPPSU
                                   403 *                                                                     *        00094000
                                   404 ** THE FLLOWING PUTITEM MACRO WILL PLACE(DATA=) THE RETRIEVED        **        00095000
                                   405 ** ITEM(NAME=) DPPSAMP2 IN THE DATA BASE .                           **        00096000
                                   406 *                                                                     *        00097000
                                   407               PUTITEM NAME=DPPSAMP2,DATA=ITEM,DCVTR=($2)                       00098000
0002D0                             408+          CNOP  0,4                                                            01-PUTIT
0002D0 4510 C2AC             002E0 409+          BAL    1,PI0040                                                      01-PUTIT
0002D4 C4D7D7E2C1D4D7F2            410+P0040     DC     CL8'DPPSAMP2'      ITEM NAME                                  01-PUTIT
0002DC 0000047E                    411+          DC     AL1(0),AL3(ITEM)                                              01-PUTIT
0002E0                             412+PI0040    CNOP  0,4                                                            01-PUTIT
0002E0 5800 C2A8             002DC 413+          L      0,P0040+8           ADDRESS OF DATA                           01-PUTIT
0002E4 58F2 0020             00020 414+          L      15,32(($2))         ADDRESS OF        CVT                     02-DPPSU
0002E8 58FF 0084             00084 415+          L      15,116+16(15)        PUTITEM  SUPPORT ROUTINE                 02-DPPSU
0002EC 45E0 C2BE             002F2 416+          BAL    14,*+6                                                        02-DPPSU
0002F0 A800                        417+          DC     AL1(168),AL1(0)                                               02-DPPSU
0002F2 BFF8 E000             00000 418+          ICM    15,8,0(14)          INSERT THE MACRO ID                       02-DPPSU
0002F6 05EF                        419+          BALR  14,15                CALL SUPPORT ROUTINE                      02-DPPSU
                                   420 ** IF ITEM DPPSAMP2 WAS UPDATED  THEN                                **        00099000
                                   421               IF F,($15),IS,ZERO,THEN                                          00100000
0002F8 12FF                        422+          LTR    $15,$15                                                       01-IF
0002FA 4770 C310             00344 423+          BC     7,IF20042                                                     01-IF
                                   424 **OUTPUT MESSAGE 68, WHICH CONTAINS MSG4(VAR=) ,TO THE  SYSTEM       **        00101000
                                   425 ** CONSOLE(ROUTE=1) .                                                **        00102000
                                   426                 MESSAGE 68,VAR=(MSG4),DCVTR=($2),ROUTE=1                       00103000
0002FE 0700                        427+          CNOP  0,4                                                            01-MESSA
000300 4510 C2E0             00314 428+          BAL    1,MSG0043                                                     01-MESSA
000304 01                          429+          DC     AL1(0+1)             VARIABLE COUNT                           01-MESSA
000305 01                          430+          DC     AL1(1)               ROUTING CODE COUNT                       01-MESSA
000306 0000                        431+          DC     AL2(0)                MESSAGE NUMBER                          01-MESSA
000308 00                          432+          DC     X'00'                ACTION CODE                              01-MESSA
000309 000000                      433+          DC     AL3(0)               USER  RETURN AREA                        01-MESSA
00030C 0000                        434+          DC     AL2(0)   ROUTING CODE                                         01-MESSA
00030E 00000000                    435+          DC     1AL4(0)             MESSAGE VARIABLE                          01-MESSA
000314                             436+MSG0043     DS   OF                                                            01-MESSA
000314 4100 0044             00044 437+          LA     0,68       LOAD MSG # INTO REGISTER 0                         01-MESSA
000318 4001 0002             00002 438+          STH    0,2(1)  MOVE MSG # TO PARAMETER LIST                          01-MESSA
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                        ASM H V 04 09.15 11/04/75

00031C 4100 0001            00001   439+          LA    0,1  LOAD ROUTING CODE INTO REGISTER 0                  01-MESSA
000320 4001 0008            00008   440+          STH   0,8(1)   STORE ROUTING CODE INTO PARAMETER LIST          01-MESSA
000324 9680 1008      00008         441+          OI    8(1),X'80'            SET HIGH BIT OF LAST ROUTINE CODE  01-MESSA
000328 1BFF                         442+          SR    15,15                 ZERO REG 15 FOR IC                 01-MESSA
00032A 43F1 0001            00001   443+          IC    15,1(1)               # OF ROUTE CODES IN PARAMETER LIST 01-MESSA
00032E 89F0 0001            00001   444+          SLL   15,1                  LENGTH OF ROUTE CODE IN PARAM LIST 01-MESSA
000332 4100 C468            0049C   445+          LA    0,MSG4                VARIABLE ADDR                      01-MESSA
000336 5001 F008            00008   446+          ST    0,8(1,15)             STORE INTO MESSAGE LIST            01-MESSA
00033A 58F2 0020            00020   447+          L     15,32(($2))          ADDRESS OF       CVT                02-DPPSU
00033E 58FF 0090            00090   448+          L     15,116+28(15)         MESSAGE  SUPPORT ROUTINE           02-DPPSU
000342 05EF                         449+          BALR  14,15                CALL SUPPORT ROUTINE                02-DPPSU
                                    450             ENDIF                                                        00104000
000344                              451+IF20042   DS    0H                                                       01-ENDIF
                                    452           ENDIF                                                          00105000
000344                              453+IF10037   DS    0H                                                       01-ENDIF
                                    454 *                                                                 *      00106000
                                    455 ** THE FOLLOWING PTIME MACRO WILL CREATE A PTQE(ADD) WHICH WILL **      00107000
                                    456 ** CAUSE DPPSAMP1(TASK= AND EP=) TO BE PATCHED THREE TIMES      **      00108000
                                    457 ** (COUNT=) WITH A 1 SECOND(START=) INTERVAL BETWEEN EACH  PATCH**      00109000
                                    458 *                                                                 *      00110000
                                    459           PTIME ADD,START=(REL,1S),COUNT=3,DCVTR=($2),EP=DPPSAMP1,      X00111000
                                                        TASK=DPPSAMP1                                            00112000
000344 4110 C318            0034C   460+          LA    1,IHB0048             SET UP PARAM LIST ADDRESS          02-PATCH
000348 47F0 C344            00378   461+          B     IHB0048A              BRANCH AROUND LIST                 02-PATCH
00034C                              462+IHB0048   DS    0F                                                       02-PATCH
00034C C4D7D7E2C1D4D7F1             463+          DC    CL8'DPPSAMP1'         TASK NAME                          02-PATCH
000354 C4D7D7E2C1D4D7F1             464+          DC    CL8'DPPSAMP1'         ENTRY POINT NAME                   02-PATCH
00035C 4040404040404040             465+          DC    CL8' '                PRTY REFERENCE NAME                02-PATCH
000364 00                           466+          DC    AL1(0)                FLAG BYTE                          02-PATCH
000365 01                           467+          DC    AL1(1)                QUEUE LENGTH                       02-PATCH
000366 0000                         468+          DC    H'0'                  PRTY RELATIVE VALUE                02-PATCH
000368 00000000                     469+          DC    A(0)                  ECB ADDRESS                        02-PATCH
00036C 0000000000000000             470+          DC    2F'0'                 FREE LENGTH,FREE ADDRESS           02-PATCH
000374 00000000                     471+          DC    A(0)                  TCBX                               02-PATCH
                            00378   472+IHB0048A  EQU   *                                                        02-PATCH
000378 41F0 C34C            00380   473+          LA    15,IHP0048            SET UP LIST ADDRESS                02-PATCH
00037C                              474+          CNOP  0,4                                                      02-PATCH
00037C 4500 C350            00384   475+          BAL   0,IHP0048A            SET UP REG 0 WITH PARAM LIST       02-PATCH
                            00380   476+IHP0048   EQU   *                                                        02-PATCH
000380 0004                         477+          DC    AL2(IHP0048A-IHP0048)  LENGTH OF PARAMS                  02-PATCH
000382 0000                         478+          DC    AL2(0)                ID                                 02-PATCH
                            00384   479+IHP0048A   EQU *                                                         02-PATCH
000384                              480+          CNOP  0,4                                                      01-PTIME
000384 5010 C36C            003A0   481+          ST    1,*+12+16             SAVE PATCH SUPERVISOR LIST ADDRESS 01-PTIME
000388 5000 C370            003A4   482+          ST    0,*+16+12             SAVE PATCH PROBLEM LIST ADDRESS    01-PTIME
00038C 4100 0004            00004   483+          LA    0,4                   REQUEST TYPE                       01-PTIME
000390 4510 C378            003AC   484+          BAL   1,PT0047ND            BRANCH AROUND PTIME PARAMETERS     01-PTIME
000394 01000064                     485+          DC    AL1(1),AL3(100)                 START TIME               01-PTIME
000398 00000000                     486+          DC    AL1(0),AL3(0)                   INTERVAL TIME            01-PTIME
00039C 08000003                     487+          DC    AL1(8),AL3(3)                   STOP TIME                01-PTIME
0003A0 00000000                     488+          DC    A(0)                  PATCH SUPERVISOR LIST              01-PTIME
0003A4 00000000                     489+          DC    A(0)                  PATCH PROBLEM PARAMETER LIST       01-PTIME
0003A8 40404040                     490+          DC    CL4'    '             PTQE ID                            01-PTIME
0003AC                              491+PT0047ND        DS 0H                                                    01-PTIME
0003AC 18F2                         492+          LR    15,($2)               CVT ADDRESS                        02-DPPSV
```

```
         DPPZSAMP         SAMPLE PROGRAM                                                                     PAGE   11

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          ASM H V 04 09.15 11/04/75

0003AE BFF8 C381            003B5   493+          ICM   15,8,*+7              ID IN HIGH ORDER BYTE OF REG          02-DPPSV
0003B2 4700 0004            00004   494+          NOP   04                    CONSTANT FOR ID                       02-DPPSV
0003B6 440F 0008            00008   495+          EX    0,8(15)               EXECUTE SVC FROM CVT                  02-DPPSV
                                    496 ** IF RETURN CODE FROM TIME MANGEMENT IS LESS THAN EIGHT THEN    **     00113000
                                    497           IF F,($15),LT,EIGHT,THEN                                          00114000
0003BA 59F0 C3F4            00428   498+          C     $15,EIGHT                                                   01-IF
0003BE 47B0 C3D4            00408   499+          BC    11,IF10050                                                  01-IF
                                    500 ** OUTPUT MESSAGE 68, WHICH CONTAINS MSG5(VAR=), TO THE SYSTEM  **      00115000
                                    501 ** CONSOLE(ROUTE=1)                                             **      00116000
                                    502              MESSAGE 68,VAR=(MSG5),DCVTR=($2),ROUTE=1                       00117000
0003C2 0700                         503+          CNOP  0,4                                                         01-MESSA
0003C4 4510 C3A4            003D8   504+          BAL   1,MSG0051                                                   01-MESSA
0003C8 01                           505+          DC    AL1(0+1)              VARIABLE COUNT                        01-MESSA
0003C9 01                           506+          DC    AL1(1)                ROUTING CODE COUNT                    01-MESSA
0003CA 0000                         507+          DC    AL2(0)                 MESSAGE NUMBER                       01-MESSA
0003CC 00                           508+          DC    X'00'                 ACTION CODE                           01-MESSA
0003CD 000000                       509+          DC    AL3(0)                USER  RETURN AREA                     01-MESSA
0003D0 0000                         510+          DC    AL2(0)   ROUTING CODE                                       01-MESSA
0003D2 00000000                     511+          DC    1AL4(0)              MESSAGE VARIABLE                       01-MESSA
0003D8                              512+MSG0051     DS  0F                                                          01-MESSA
0003D8 4100 0044            00044   513+          LA    0,68      LOAD MSG # INTO REGISTER 0                        01-MESSA
0003DC 4001 0002            00002   514+          STH   0,2(1)  MOVE MSG # TO PARAMETER LIST                        01-MESSA
0003E0 4100 0001            00001   515+          LA    0,1  LOAD ROUTING CODE INTO REGISTER 0                      01-MESSA
0003E4 4001 0008            00008   516+          STH   0,8(1)   STORE ROUTING CODE INTO PARAMETER LIST             01-MESSA
0003E8 9680 1008      00008         517+          OI    8(1),X'80'            SET HIGH BIT OF LAST ROUTINE CODE     01-MESSA
0003EC 18FF                         518+          SR    15,15                 ZERO REG 15 FOR IC                    01-MESSA
0003EE 43F1 0001            00001   519+          IC    15,1(1)               # OF ROUTE CODES IN PARAMETER LIST    01-MESSA
0003F2 89F0 0001            00001   520+          SLL   15,1                  LENGTH OF ROUTE CODE IN PARAM LIST    01-MESSA
0003F6 4100 C470            004A4   521+          LA    0,MSG5                VARIABLE ADDR                         01-MESSA
0003FA 5001 F008            00008   522+          ST    0,8(1,15)             STORE INTO MESSAGE LIST               01-MESSA
0003FE 58F2 0020            00020   523+          L     15,32(($2))          ADDRESS OF        CVT                  02-DPPSU
000402 58FF 0090            00090   524+          L     15,116+28(15)         MESSAGE  SUPPORT ROUTINE              02-DPPSU
000406 05EF                         525+          BALR  14,15                CALL SUPPORT ROUTINE                   02-DPPSU
                                    526           ENDIF                                                             00118000
000408                              527+IF10050   DS    0H                                                          01-ENDIF
                                    528 *                                                               *           00119000
                                    529 ** THE EXIT MACRO WILL RESTORE ALL REGISTERS AS THEY  WERE WHEN **          00120000
                                    530 ** DPPZSAMP WAS ENTERED AND RETURN BACK TO THE SYSTEM.           **          00121000
                                    531 *                                                               *           00122000
                                    532           EXIT  CODE=0                                                      00123000
000408                              533+          DS    0H                                                          01-EXIT
000408 181D                         534+          LR    1,13 .         SUBPOOL ADDRESS                              01-EXIT
00040A 58DD 0004            00004   535+          L     13,4(13) .     GET CALLER'S SAVE AREA                       01-EXIT
00040E 5800 C000            00034   536+          L     0,TKG0001G     .   LOAD SP AND LV PARAMETERS                01-EXIT
                                    537+*      FREEMAIN R,LV=(0),A=(1)
000412 4111 0000            00000   538+          LA    1,0(1,0)                     CLEAR THE HIGH ORDER BYTE XM4571 02-FREEM
000416 0A0A                         539+          SVC   10                           ISSUE FREEMAIN SVC        P2504  02-FREEM
000418 98EC D00C            0000C   540+          LM    14,12,12(13)                           RESTORE THE REGISTERS  02-RETUR
00041C 92FF D00C      0000C         541+          MVI   12(13),X'FF'                           SET RETURN INDICATION  02-RETUR
000420 41F0 0000            00000   542+          LA    15,0(0,0)                              LOAD RETURN CODE       02-RETUR
000424 07FE                         543+          BR    14                                     RETURN                 02-RETUR
                                    544 *                                                               *           00124000
                                    545 ** SRTOS SAMPLE PROGRAM DATA AND CONSTANT AREA                  **          00125000
                                    546 *                                                               *           00126000

000426 0000
```

```
        DPPZSAMP        SAMPLE PROGRAM                                                                           PAGE   12

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                         ASM H V 04 09.15 11/04/75

000428 00000008                     547 EIGHT    DC     F'8'                 CONSTANT OF 8 USED IN IF INSTRUCTION 00126100
000430                              548 ARRAY    DS     D                    ADDRESS OF ARRAY DPPZSAMP            00127000
000438                              549 LOGCOPY  DS     CL70                 LOGGED COPY OF ARRAY DPPZSAMP        00128000
00047E                              550 ITEM     DS     CL6                  CONTENTS OF ITEM DPPSAMP2            00129000
                                    551 *                                                               *     00130000
                                    552 **       SAMPLE PROGRAM DIAGNOSTIC MESSAGES  VARIABLES          **    00131000
                                    553 *                                                               *     00132000
000484 D7C1E3C3C8404040             554 MSG1     DC     CL8'PATCH'           PATCH MACRO DIAGNOSTIC MESSAGE       00133000
00048C D7E4E3D3D6C74040             555 MSG2     DC     CL8'PUTLOG'          PUTLOG   MACRO DIAGNOSTIC MESSAGE    00134000
000494 D7E4E3C1D9D9C1E8             556 MSG3     DC     CL8'PUTARRAY'        PUTARRAY MACRO DIAGNOSTIC MESSAGE    00135000
00049C D7E4E3C9E3C5D440             557 MSG4     DC     CL8'PUTITEM'         PUTITEM MACRO DIAGNOSTIC MESSAGE     00136000
0004A4 D7E3C9D4C5404040             558 MSG5     DC     CL8'PTIME'           PTIME MACRO DIAGNOSITC MESSAGE       00137000
                                    559          END                                                          00138000



                                                                   RELOCATION DICTIONARY                         PAGE   13

 POS.ID   REL.ID   FLAGS   ADDRESS                                                                 ASM H V 04 09.15 11/04/75

  0001     0001     08     000195
  0001     0001     0C     000198
  0001     0001     08     000269
  0001     0001     08     0002DD
```

```
                                        CROSS REFERENCE                                                                PAGE   14

SYMBOL     LEN     VALUE    DEFN    REFERENCES                                                                 ASM H V 04 09.15 11/04/75

$1         00001 00000001   0058    0113
$15        00001 0000000F   0072    0148  0148  0197  0197  0235  0235  0276  0276  0323  0323  0375  0375  0422  0422  0498
$2         00001 00000002   0059    0113  0142  0173  0189  0218  0227  0256  0271  0302  0315  0344  0367  0400  0414  0447
                                    0492  0523
$3         00001 00000003   0060    0199  0217
$4         00001 00000004   0061    0280  0301  0314
ARRAY      00008   000430   0548    0188  0199
DPPZSAMP   00001 00000000   0054    0096
EIGHT      00004   000428   0547    0498
GA0011     00001   0000E0   0187    0185
GI0035     00001   00026C   0365    0362
IF10007    00002   0000D4   0177    0149
IF10013    00002   00025C   0354    0198
IF10037    00002   000344   0453    0376
IF10050    00002   000408   0527    0499
IF20018    00002   00025C   0352    0236
IF20042    00002   000344   0451    0423
IF30023    00002   00025C   0350    0277
IF40028    00002   00025C   0348    0324
IHB0005    00004   000044   0124    0122
IHB0005A   00001 00000070   0134    0123
IHB0048    00004   00034C   0462    0460
IHB0048A   00001 00000378   0472    0461
IHP0005    00001 00000078   0138    0135  0139
IHP0005A   00001 0000007C   0141    0137  0139
IHP0048    00001 00000380   0476    0473  0477
IHP0048A   00001 00000384   0479    0475  0477
ITEM       00006   00047E   0550    0364  0398  0411
I0035      00008   000260   0363    0366
LOGCOPY    00070   000438   0549    0267  0280
MSG0008    00004   0000A4   0162    0154
MSG0014    00004   000114   0211    0204
MSG0019    00004   00016C   0248    0241
MSG0024    00004   0001D4   0292    0284
MSG0029    00004   000238   0336    0329
MSG0038    00004   0002A0   0389    0381
MSG0043    00004   000314   0436    0428
MSG0051    00004   0003D8   0512    0504
MSG1       00008   000484   0554    0171
MSG2       00008   00048C   0555    0254
MSG3       00008   000494   0556    0342
MSG4       00008   00049C   0557    0445
MSG5       00008   0004A4   0558    0521
PA0026     00001   00020C   0313    0311
PI0040     00001   0002E0   0412    0409
PT0047ND   00002   0003AC   0491    0484
P0040      00008   0002D4   0410    0413
TKG0001G   00001   000034   0102    0084  0536
TKG0001M   00004   000034   0101    0104
WORK       00001 00000000   0094    0097
```

```
                                   DIAGNOSTIC CROSS REFERENCE AND ASSEMBLER SUMMARY                    PAGE   15

                                                                                     ASM H V 04 09.15 11/04/75
  NO STATEMENTS FLAGGED IN THIS ASSFMBLY

OVERRIDING PARAMETERS-  NODECK,NOLOAD,XREF(SHORT)
OPTIONS FOR THIS ASSEMBLY
 NODECK, NOOBJECT, LIST, XREF(SHORT), NORENT, NOTEST, NOBATCH, ALIGN, ESD, RLD, LINECOUNT(55), FLAG(0), SYSPARM()
NO OVERRIDING DD NAMES

  165 CARDS FROM SYSIN    4267 CARDS FRCM SYSLIB
  654 LINES OUTPUT           0 CARDS OUTPUT
```

```
                                   EXTERNAL SYMBOL DICTIONARY                                        PAGE    1

SYMBOL   TYPE  ID  ADDR  LENGTH  LD ID                                                    ASM H V 04 09.15 11/04/75

DPPSAMP1  SD 0001 000000 0000C9
```

```
         DPPSAMP1 - SAMPLE PROGRAM PATCH ENTRY ROUTINE                                         PAGE.   2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  ASM H V 04 09.15 11/04/75

                                    18 ********************************************************************** 00002000
                                    19 * MODULE NAME = DPPSAMP1                                             * 00002100
                                    20 * DESCRIPTIVE NAME = SAMPLE PROGRAM PATCH ENTRY ROUTINE              * 00002200
                                    21 * FUNCTION = DPPSAMP1 FUNCTION IS TO BE PATCHED BY THE SPECIAL REAL  * 00003000
                                    22 *      TIME OPERATING SYSTEM SAMPLE PROGRAM(DPPZSAMP)                * 00003100
                                    23 * NOTES = THE PROGRAM IS ENTERED FOUR TIMES. ONE TIME BY A PATCH MACRO* 00003200
                                    24 *      ISSUED BY DPPZSAMP AND THREE 1 SECOND CYCLIC PATCHES ISSUED BY * 00004000
                                    25 *      A PTIME MACRO IN DPPZSAMP.                                    * 00004100
                                    26 * DEPENDENCIES = NONE                                                * 00004200
                                    27 * RESTRICTIONS = NONE                                                * 00005000
                                    28 * REGISTER CONVENTIONS = ALL REGS ARE ASSIGNED AS $R WHERE REGS 0-15 * 00005100
                                    29 *      ARE $0-$15                                                    * 00005200
                                    30 * MODULE TYPE = SAMPLE PROGRAM                                       * 00006000
                                    31 *      PROCESSOR = ASSEMBLER F                                       * 00006100
                                    32 *      MODULE SIZE = 208 DECIMAL BYTES                               * 00006200
                                    33 *      ATTRIBUTES = REENTRANT                                        * 00007000
                                    34 * ENTRY POINT = DPPSAMP1                                             * 00007100
                                    35 * INPUT = NONE                                                       * 00007200
                                    36 * OUTPUT = SPECIAL REAL TIME OPERATING SYSTEM MESSAGE 66             * 00008000
                                    37 * RETURN = NORMAL OS/VS RETURN VIA BR14. NO RETURN CODES             * 00008100
                                    38 * EXTERNAL REFERENCES = NONE                                         * 00009000
                                    39 * MACROS = BEGIN EXIT MESSAGE                                        * 00010000
                                    40 ********************************************************************** 00011000


                                    42 ***                                                                    00013000
                                    43 *** THE BEGIN MACRO WILL ESTABLISH A BASE REGISTER FOR DPPSAMP1        00014000
                                    44 ***                                                                    00015000
                                    45          BEGIN DPPSAMP1,BASE=(12),SAVEA=(GETMAIN,WORK),LV=72           00016000
000000                              46+DPPSAMP1 CSECT .                'MAIN' CONTROL SECTION                 01-BEGIN

                                    48+*                               GOES THRU REGISTER EQUATE ONLY ONCE
                      00000         49+$0      EQU    0 ?              **                                     02-EQUAT
                      00001         50+$1      EQU    1 ?              **                                     02-EQUAT
                      00002         51+$2      EQU    2 ?              **                                     02-EQUAT
                      00003         52+$3      EQU    3 ?              **                                     02-EQUAT
                      00004         53+$4      EQU    4 ?              **                                     02-EQUAT
```

```
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          ASM H V 04 09.15 11/04/75
                           00005    54+$5       EQU   5 ?            **                                      02-EQUAT
                           00006    55+$6       EQU   6 ?            **IF THESE SUBSTITUTES ARE USED AS      02-EQUAT
                           00007    56+$7       EQU   7 ?            **REGISTER NUMBERS THE CROSS-REFERENCE  02-EQUAT
                           00008    57+$8       EQU   8 ?            **TABLE WILL PROVIDE A LIST OF WHERE    02-EQUAT
                           00009    58+$9       EQU   9 ?            **EACH REGISTER WAS USED                02-EQUAT
                           0000A    59+$10      EQU   10 ?           **                                      02-EQUAT
                           0000B    60+$11      EQU   11 ?           **                                      02-EQUAT
                           0000C    61+$12      EQU   12 ?           **                                      02-EQUAT
                           0000D    62+$13      EQU   13 ?           **                                      02-EQUAT
                           0000E    63+$14      EQU   14 ?           **                                      02-EQUAT
                           0000F    64+$15      EQU   15 ?           **                                      02-EQUAT
                           00000    65+FPR0     EQU   0                                                      02-EQUAT
                           00002    66+FPR2     EQU   2                                                      02-EQUAT
                           00004    67+FPR4     EQU   4                                                      02-EQUAT
                           00006    68+FPR6     EQU   6                                                      02-EQUAT

000000                              70+         DS    0D .           FOR BOUNDARY ALIGNMENT                  01-BEGIN
                           00000    71+         USING *,15 .         TEMPORARY BASE DECLARATION              01-BEGIN
000000 47F0 F00E           0000E    72+         B     14(0,15)                          BRANCH AROUND ID     02-SAVE
000004 08                           73+         DC    AL1(8)                            LENGTH OF IDENTIFIER 02-SAVE
000005 C4D7D7E2C1D4D7F1             74+         DC    CL8'DPPSAMP1'                     IDENTIFIER           02-SAVE
00000D 00
00000E 90EC D00C           0000C    75+         STM   14,12,12(13)                      SAVE REGISTERS       02-SAVE
000012 5800 F034           00034    76+         L     0,TKG0001G       .  LOAD SP AND LV PARAMETERS          01-BEGIN
                                    77+*        GETMAIN R,LV=(0)
000016 4510 F01A           0001A    78+         BAL   1,*+4                             INDICATE GETMAIN     02-GETMA
00001A 0A0A                         79+         SVC   10                                ISSUE GETMAIN SVC    02-GETMA
00001C 50D1 0004           00004    80+         ST    13,4(1) .      SAVE CALLER'S SAVE AREA POINTER         01-BEGIN
000020 501D 0008           00008    81+         ST    1,8(13) .      FOR DOWNWARD SAVE AREA TRACE            01-BEGIN
000024 18D1                         82+         LR    13,1 .         ESTABLISH OWN SAVE AREA POINTER         01-BEGIN
000026 581D 0004           00004    83+         L     1,4(13) .           RESTORE 15,0,1                     01-BEGIN
00002A 98E1 100C           0000C    84+         LM    14,1,12(1)          RESTORE GET REGS                   01-BEGIN

000000                              86+WORK      DSECT .          BEGIN GETMAINED AREA                    01-BEGIN
000000                              87+         DS    9D .           OWN SAVE AREA                           01-BEGIN
00002E                              88+DPPSAMP1 CSECT                                                    01-BEGIN
                           00000    89+         USING WORK,13                                                01-BEGIN

00002E 0700                         91+         CNOP  0,4                                                    01-BEGIN
000030 45C0 F038           00038    92+         BAL   12,*+8      .       ESTABLISH INITIAL 'MAIN' CSECT BASE  01-BEGIN
000034                              93+TKG0001M DS    0F .           BASE REFERENCE                          01-BEGIN
000034 00000048                     94+TKG0001G    DC    AL1(0),AL3(72)     . SUBPOOL, LENGTH                01-BEGIN
                                    95+         DROP  15                                                     01-BEGIN
                           00034    96+         USING TKG0001M,12                                            01-BEGIN
000038 5821 0000           00000    97          L   $2,0($1)                ADDRESS OF       XCVT            00017000
                                    98          MESSAGE 66,VAR=(MSG),DCVTR=($2)  ISSUE MESSAGE               00018000
00003C                              99+         CNOP  0,4                                                    01-MESSA
00003C 4510 C018           0004C   100+         BAL   1,MSG0005                                              01-MESSA
000040 01                          101+         DC    AL1(0+1)                 VARIABLE COUNT                01-MESSA
000041 00                          102+         DC    AL1(0)                   ROUTING CODE COUNT            01-MESSA
000042 0000                        103+         DC    AL2(0)                    MESSAGE NUMBER               01-MESSA
000044 00                          104+         DC    X'00'                    ACTION CODE                   01-MESSA
000045 000000                      105+         DC    AL3(0)                   USER  RETURN AREA             01-MESSA
000048 00000000                    106+         DC    1AL4(0)                 MESSAGE VARIABLE               01-MESSA
```

```
         DPPSAMP1 - SAMPLE PROGRAM PATCH ENTRY ROUTINE                                                        PAGE    4

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM H V 04 09.15 11/04/75

00004C                              107+MSG0005    DS  0F                                                       01-MESSA
00004C 4100 0042            00042   108+         LA    0,66      LOAD MSG # INTO REGISTER 0                    01-MESSA
000050 4001 0002            00002   109+         STH   0,2(1)  MOVE MSG # TO PARAMETER LIST                    01-MESSA
000054 18FF                         110+         SR    15,15              ZERO REG 15 FOR IC                   01-MESSA
000056 43F1 0001            00001   111+         IC    15,1(1)            # OF ROUTE CODES IN PARAMETER LIST   01-MESSA
00005A 89F0 0001            00001   112+         SLL   15,1               LENGTH OF ROUTE CODE IN PARAM LIST   01-MESSA
00005E 4100 C056            0008A   113+         LA    0,MSG              VARIABLE ADDR                        01-MESSA
000062 5001 F008            00008   114+         ST    0,8(1,15)          STORE INTO MESSAGE LIST              01-MESSA
000066 58F2 0020            00020   115+         L     15,32(($2))       ADDRESS OF       CVT                  02-DPPSU
00006A 58FF 0090            00090   116+         L     15,116+28(15)      MESSAGE  SUPPORT ROUTINE             02-DPPSU
00006E 05EF                         117+         BALR  14,15             CALL SUPPORT ROUTINE                  02-DPPSU
                                    118 ***                                                                    00020000
                                    119 *** THE EXIT MACRO WILL RESTORE ALL REGISTERS AS THEY WERE WHEN ***    00021000
                                    120 *** DPPSAMP1 WAS ENTERED AND WILL RETURN BACK TO THE SYSTEM            00022000
                                    121 ***                                                                    00023000
                                    122          EXIT                                                          00023100
000070                              123+         DS    0H                                                      01-EXIT
000070 181D                         124+         LR    1,13 .        SUBPOOL ADDRESS                           01-EXIT
000072 58DD 0004            00004   125+         L     13,4(13) .    GET CALLER'S SAVE AREA                    01-EXIT
000076 5800 C000            00034   126+         L     0,TKG0001G    .   LOAD SP AND LV PARAMETERS             01-EXIT
                                    127+*     FREEMAIN R,LV=(0),A=(1)
00007A 4111 0000            00000   128+         LA    1,0(1,0)                 CLEAR THE HIGH ORDER BYTE XM4571 02-FREEM
00007E 0A0A                         129+         SVC   10                       ISSUE FREEMAIN SVC        P2504  02-FREEM
000080 98EC D00C            0000C   130+         LM    14,12,12(13)                       RESTORE THE REGISTERS  02-RETUR
000084 92FF D00C      0000C         131+         MVI   12(13),X'FF'                       SET RETURN INDICATION  02-RETUR
000088 07FE                         132+         BR    14                                 RETURN                 02-RETUR
00008A E3C1E2D2406040C4             133 MSG      DC    CL63'TASK - DPPSAMP1 WAS ENTERED AT ENTRY POINT - DPPSAMX00024000
000092 D7D7E2C1D4D7F140                                P1'                                                     00025000
                                    134          END                                                           00026000
```

```
                                             CROSS REFERENCE                                                  PAGE    5

SYMBOL    LEN    VALUE   DEFN   REFERENCES                                                    ASM H V 04 09.15 11/04/75

$1        00001 00000001  0050   0097
$2        00001 00000002  0051   0097  0115
DPPSAMP1  00001 00000000  0046   0088
MSG       00063   00008A  0133   0113
MSG0005   00004   00004C  0107   0100
TKG0001G  00001   000034  0094   0076  0126
TKG0001M  00004   000034  0093   0096
WORK      00001 00000000  0086   0089
```

```
                         DIAGNOSTIC CROSS REFERENCE AND ASSEMBLER SUMMARY                         PAGE    6

                                                                                  ASM H V 04 09.15 11/04/75
  NO STATEMENTS FLAGGED IN THIS ASSEMBLY

OVERRIDING PARAMETERS-  NODECK,NOLOAD,XREF(SHORT)
OPTIONS FOR THIS ASSEMBLY
 NODECK, NOOBJECT, LIST, XREF(SHORT), NORENT, NOTEST, NOBATCH, ALIGN, ESD, RLD, LINECOUNT(55), FLAG(0), SYSPARM()
NO OVERRIDING DD NAMES

   40 CARDS FROM SYSIN     1508 CARDS FROM SYSLIB
  163 LINES OUTPUT            0 CARDS OUTPUT
```

APPENDIX B. LISTING AIDS

Much of the source code for the Special Real Time Operating System is written in OS/VS1 assembler language using structured programming techniques. The structured programming vehicles for the Special Real Time Operating System are a set of macro instructions know as HLAL. HLAL is not a part of the Special Real Time Operating System PRPQ, but is distributed with it as a necessary aid in assembling most of the modules of the Special Real Time Operating System.

Most assembler language programs written in structured code are easier to read if the nesting level of the various statements is printed along with the listing Also, the various statements should be indented to show the nesting level in a graphic manner. Nesting level refers to a statement being in a basis IF/THEN/ELSE structure, or structures within that structure.

Two listing aid programs are provided with the Special Real Time Operating System PRPQ to facilitate the showing of the nesting level of the Special Real Time Operating System source modules. One of these programs, PPLPTPCH, post-processes the IEBPTPCH listing of a module; the other, PPLUPDTE, post-processes the IEBUPDTE listing of a module. Each program shifts the print line of its input to produce a structured listing. It does not alter (shift) the columns in which the source is actually resident in the source partitioned data set. It will automatically shift each member whose first card image does not contain the operation code of MACRO.

Example 1 depicts the JCL which could be used to run both the PPLPTPCH and PPLUPDTE utility programs. Example 2 shcws sample output from both programs. Example 3 shows the output from IEBUPDTE and IEBPTPCH as it would appear if the post-processor were not used. IEBUPDTE and IEBPTPCH are described in the SRL: OS/VS1 Utilities, GC35-0005.

PPLUPDTE and PPLPTPCH are contained in data set A5799AHE.OBJECT.

```
//JOB 1          JOB
//A  EXEC        PGM=1EBPTPCH
//SYSPRINT   DD  DUMMY
//SYSUT1     DD  DSN=SOURCEDS, DISP=SHR
//SYSUT2     DD  UNIT=SYSDA, SPACE=(CYL,(1)), DISP=(NEW,PASS),
//               DCB=(RECFM=FBA, LRECL=121, BLKSIZE=3630)
//SYSIN      DD  *
     PRINT       TYPORG=PO, MAXNAM=10, MAXFLDS=10
     MEMBER      NAME=MEM1
     RECORD      FIELD=(80)
//B  EXEC        PGM=PPLPTPCH
//SYSUT1     DD  DSN=*.A.SYSUT2,DISP=(OLD,DELETE)
//SYSUT2     DD  SYSOUT=A
//STEPLIB    DD  DSN=A5799AHE,OBJECT,DISP=SHR
//C  EXEC        PGM=1EBUPDTE
//SYSPRINT   DD  UNIT=SYSDA,SPACE=(CYL,(1,1,)),DISP=(NEW,PASS),
//               DCB=(RECFM=FBA,LRECL=121, BLKSIZE=3630)
//SYSUT1     DD  DSN=SOURCEDS, DISP=SHR
//SYSUT2     DD  DSN=SOURCEDS, DISP=OLD
//SYSIN      DD  *
 /       CHANGE  LIST=ALL,NAME=MEM1
                 ST       $6,GORP  SAVE REG                             01000000
GORP         DS  F                                                      11000000
//D  EXEC        PGM=PPLUPDTE
//SYSUT1     DD  DSN=*.C.SYSPRINT, DISP=(OLD,PASS)
//SYSUT2     DD  SYSOUT=A
//STEPLIB    DD  DSN=A5799AHE.OBJECT, DISP=SHR
```

EXAMPLE 1

```
                                                                      PAGE 0001
MEMBER NAME  MEM1
   * SAMPLE MODULE TO ILLUSTRATE INDENTION                              01000000
   IF    C,ONE,EQ,TWO,THEN                                              02000000
01   * INDENTED COMMENT CAUSED BY ABOVE                                 03000000
01   UNTIL (B,LOC,NE,3)OR                                               04000000
01   UNTIL (F,($6),EQ,X),DO                                             05000000
02     STH   7,LOC                                                      06000000
02     L     ETC *****                                                  07000000
01   ENDDO                                                              08000000
01   * NOTE INDENTION MOVES BACK                                        09000000
   ENDIF                                                                10000000
   END                                                                  12000000
```

EXAMPLE 2

```
                                                                       PAGE 0001
MEMBER NAME  MEM1
* SAMPLE MODULE TO ILLUSTRATE INDENTION                                  01000000
         IF    C,ONE,EQ,TWO,THEN                                         02000000
* INDENTED COMMENT CAUSED BY ABOVE                                       03000000
 UNTIL (B,LOC,NE,3),OR                                                   04000000
               UNTIL (F,($6),EQ,X),DO                                    05000000
         STH   7,LOC                                                     06000000
         L     ETC *****                                                 07000000
 ENDDO                                                                   08000000
* NOTE INDENTION MOVES BACK                                              09000000
               ENDIF                                                     10000000
         END                                                             12000000


       SYSIN                            NEW MASTER       IEBUPDTE LOG PAGE 0001
       ./ CHANGE LIST=ALL,NAME=MEM1
IEB826I MEMBER NAME FOUND IN OM DIRECTORY AS AN ALIAS-
   * SAMPLE MODULE TO ILLUSTRATE INDENTION                               01000000
   ST    $6,GORP             SAVE REG                                    01100000
   IF    C,ONE,EQ,TWO,THEN                                               02000000
01   * INDENTED COMMENT CAUSED BY ABOVE                                  03000000
01   UNTIL (B,LOC,NE,3),OR                                               04000000
01   UNTIL (F,($6),EQ,X),DO                                              05000000
02     STH   7,LOC                                                       06000000
02     L     ETC *****                                                   07000000
01   ENDDO                                                               08000000
01   * NOTE INDENTION MOVES BACK                                         09000000
   ENDIF                                                                 10000000
   GORP     DS    F                                                      11000000
   END                                                                   12000000
IEB816I MEMBER NAME (MEM1     ) FOUND IN NM DIRECTORY. TTR IS NOW ALTERED
IEB818I HIGHEST CONDITION CODE WAS 00000000
IEB819I END OF JOB IEBUPDTE.
```

EXAMPLE 3

```
       SYSIN                             NEW MASTER       IEBUPDTE LOG PAGE 0001
       ./ CHANGE LIST=ALL,NAME=MEM1
IEB826I MEMBER NAME FOUND IN OM DIRECTORY AS AN ALIAS-
CHANGED TO TRUE NAME IN NM DIRECTORY.
     * SAMPLE MODULE TO ILLUSTRATE INDENTION                            01000000
              ST    $6,GORP              SAVE REG                       01100000
              IF    C,ONE,EQ,TWO,THEN                                   02000000
     * INDENTED COMMENT CAUSED BY ABOVE                                 03000000
      UNTIL (B,LOC,NE,3),OR                                             04000000
                    UNTIL (F,($6),EQ,X),DO                              05000000
              STH   7,LOC                                               06000000
              L     ETC *****                                           07000000
      ENDDO                                                             08000000
     * NOTE INDENTION MOVES BACK                                        09000000
                    ENDIF                                               10000000
     GORP     DS    F                                                   11000000
              END                                                       12000000
IEB816I MEMBER NAME (MEM1    ) FOUND IN NM DIRECTORY. TTR IS NOW ALTERED.
IEB818I HIGHEST CONDITION CODE WAS 00000000
IEB819I END OF JOB IEBUPDTE.
```

APPENDIX C. MODULE NAME - FUNCTION CROSS REFERENCE

Module Name	
DOMICEXT	SUBSTITUTE EXTERNAL FIRST LEVEL INTERRUPT HANDLER
DOMIRBT	FAILOVER/RESTART BOOTSTRAP AND PROBE
DOMIRCMN	CONTINUOUS MONITOR
DOMIRCPY	COPY A FAILOVER/RESTART DATA SET
DOMIRFLV	LOAD 1 F/R SVC
DOMIRFL2	LOAD 2 F/R SVC
DOMIRINT	F/R-EXTERNAL INTERRUPT INIT.
DOMIRNIP	RE-NIP
DOMIRPRB	PROBE
DOMIRWT	FAILOVER/RESTART WRITE
	THE FOLLOWING 3 MODULES ARE NAMED AT SYSGEN TIME ACCORDING TO NUMBERS SUPPLIED
DOMISVC1	TYPE 1 SVC PREFIX HANDLER
DOMISVC2	TYPE 2 SVC PREFIX HANDLER
DOMISVC4	TYPE 4 SVC PREFIX HANDLER
DOMXLIST	PREPARE IEHLIST INPUT
DOMXSTG1	STAGE 1 OF SYSGEN UTILITY
DPPCALCF	CALCULATE TIME CORRECTION FACTOR
DPPCPTIM	PTIME MONITOR ROUTINE
DPPCTIME	TIME UPDATE ROUTINE
DPPCTIME2	ALTERNATE TIME UPDATE ROUTINE
DPPCTSVC	PTIME TYPE 2 SVC
DPPCUPCF	UPDATE TIME CORRECTION FACTOR
DPPDARAY	GET/PUTARRAY PROCESSOR
DPPDBLOK	GET/PUT BLOCK SUBROUTINE
DPPDBSIF	DATA BASE SLAVE PARTITION INTERFACE
DPPDFREQ	CYCLIC LOGGING ROUTINE
DPPDGETL	GETLOG ROUTINE
DPPDITEM	GET/PUTITEM PROCESSOR
DPPDPUTL	PUTLOG ROUTINE
DPPDRIFE	DUMMY INIT. TIME SETTER
DPPDRIFT	TIME DRIFT CORRECTION
DPPDSUB2	SLAVE PARTITION INTERFACE ROUTINE
DPPDUMPL	DUMPLOG ROUTINE
DPPDUPDL	LOGGING REFRESH ROUTINE
DPPDWRST	DATA BASE OPEN/CLOSE FOR RESTART
DPPFAONC	FORTRAN SUBROUTINE FOR COPY/ADDR/BIT SET
DPPFIXFR	PAGE FIX/FREE HANDLER
DPPIDBAS	DATA BASE INITIALIZATION
DBPIIRP	SCHEDULE IRB FOR DPPDWRST
DPPILOGN	LOGGING INITIALIZATION
DPPINIT	SYSTEM INITIALIZATION
DPPINIT1	INITIALIZATION SUBSYSTEM PATCHOR
DPPILOGN	LOGGING INITIALIZATION
DPPIPFIX	PAGE FIX ROUTINE
DPPIPFRE	PAGE FREE/UNFIX ROUTINE
DPPISTAE	JOB STEP TASK STAE ROUTINE
DPPITIMI	TIME INITIALIZATION
DPPMINIT	MSG HANDLER INITIALIZATION
DPPMMSG	SYSTEM MESSAGE FORMATTER
DPPMMSGV	SYSTEM MESSAGE ROUTING CODE CHANGE
DPPMMSG1	SYSTEM MESSAGE OUTPUT ROUTINE
DPPPARM	PL/I AND FORTRAN PARAMETER INTERFACE ROUTINE

DPPPIF	PL/I AND FORTRAN INTERFACE ROUTINE
DPPSASOC	DDS ASYNCHRONIS OPEN OR CLOSE
DPPSBF1	BLDL FIND TYPE D FOR A DDS
DPPSBFST	BLDL FIND TYPE D STOW FOR A DDS
DPPSCHCK	DDS CHECK MODULE
DPPSCHPR	SET A PRIMARY DECB AND A BACKUP DECB
DPPSCL1	CLOSE A DDS
DPPSCLUP	DDS CLEAN UP ROUTINE
DPPSCMPR	COMPARE FOR DDS
DPPSCRBK	CREATE A DDS BACKUP
DPPSCT2T	COPY TRACK TO TRACK
DPPSINIT	INITIALIZE THE DDS SYSTEM
DPPSLOCK	LOCK A DDS
DPPSMSGI	DDS INPUT MESSAGE PROCESSOR
DPPSMSGO	DDS MESSSAGE OUTPUT PROCESSOR
DPPSNOTE	PERFORM NOTE ON A DDS
DPPSNTPT	BRANCH CODE FOR NOTE POINT
DPPSOP1	OPEN A DDSDCB
DPPSOPCL	OPEN CLOSE HALF OF A DDS
DPPSPNTF	PERFORM POINT FIND TYPE C ON A DDS
DPPSRCIO	RECREATE I O FOR A DDS HALF
DPPSRDWT	READ WRITE MODULE FOR DDS
DPPSRSTR	DDS FAILOVER/RESTART
DPPSSHAR	SHARE A DDS
DPPSSRCH	SEARCH A FIXED LENGTH TABLE FOR AN ENTRY WHO
DPPSST1	STOW FOR A DDS
DPPSSWCH	SWITCH PRIMARY TO BACKUP FOR A DDS
DPPSTBOS	TAKE A BACKUP OUT OF SERVICE
DPPSUNLK	UNLOCK A DDS
DPPSUNSH	UNSHARE A DDS
DPPSWRST	DDS WRITE STATUS
DPPSXTCB	LOCATE ALLOCATE A DDSXTCBC FOR AN INPUT TASK
DPPTCBGT	CBGET TYPE 1 SVC ROUTINE
DPPTCSVC	CHAIN TYPE 1 SVC ROUTINE
DPPTDLMP	LOAD MODULE PURGE
DPPTDSVC	DPATCH SVC RTN
DPPTETXR	END OF TASK EXIT
DPPTGWFW	GETWA/FREEWA INTERFACE
DPPTIMPS	STAE INITIALIZATION
DPPTPMON	PATCH MONITOR
DPPTPSVC	PATCH SVC RTN
DPPTPWQE	PURGEWQ ROUTINE
DPPTQIMP	QS IMP COMMAND PROCESSOR
DPPTRSVC	REPATCH SVC RTN
DPPTSMON	SYSTEM MONITOR
DPPTSTAE	STAE DUMP/NODUMP ROUTINE
DPPTWAIT	PURGEWQ WAIT ROUTINE
DPPTWQDL	WQE DELETE RTN
DPPTWSVC	GETWA-FREEWA TYPE 1 SVC ROUTINE
DPPUMSG	SYSTEM MESSAGE OFFLINE PROCESSOR
DPPXDBAS	DATA BASE FINAL PHASE PROCESSOR, FIRST LOAD
DPPXDBAT	DATA BASE FINAL PHASE PROCESSOR, SECOND LOAD
DPPXDBCP	DATA BASE BDAM DATA SET COMPRESS
DPPXDBDA	DATA BASE FINAL PHASE PROCESSOR SUPPORT ROUTINE TO WRITE DATA TO DATA BASE BDAM DATA SETS
DPPXDBIN	OFFLINE DATA BASE TABLE CONSTRUCT
DPPXDBLG	DATA BASE FINAL PHASE PROCESSOR SUPPORT ROUTINE TO CALCULATE LOGGING ARRAY BLOCK COUNT AND BLOCK SIZE
DPPXDBPT	DATA BASE PRINT UTILITY
DPPXDEFL	DEFINE LOCK ROUTINE
DDPPXDPB	DATA RECORDING AND PLAYBACK
DPPXDRC	DATA RECORDING COLLECTION ROUTINE
DPPXDRCX	DUMMY DATA RECORDING COLLECTION

DPPXIMPP	INPUT MESSAGE PROCESSOR
DPPXIMPW	INPUT MESSAGE PROCESSOR WTOR ROUTINE
DPPXKILL	ORDERLY TERMINATION ROUTINE
DPPXLOCK	LOCK ROUTINE
DPPXNRTI	DATA PLAYBACK OFFLINE ENTRY ROUTINE
DPPXPCON	PLAYBACK REQUEST INTERPRETER
DPPXRDR	DATA PLAYBACK PRINT ROUTINE
DPPXRINT	DATA RECORDING INITIALIZATION
DPPXRPRT	REPORT DATA OUTPUT PROCESSOR
DPPXS2SC	LOCATE INSERT CARDS IN OS/VS1 STAGE II SYSGEN DECK
DPPXSVCP	SETPSW TYPE 1 SVC
DPPXUTIL	OFFLINE UTILITY CONTROL PROGRAM
IEAXYZ5	ALTERNATE NAME FOR DOMICEXT

APPENDIX D. SPECIAL REAL TIME OPERATING SYSTEM PROGRAMS/MACROS

Below are programs/macros that comprise the Special Real Time Operating System program package. Macros are noted with asterisks.

Basic Source Programs/Macros

ARRAY	*
ARRAYDEF	
BEGIN	*
BGNSEG	*
BIT	*
BLOCK	*
BLOCKDEF	
BYTE	*
CASE	*
CBFREE	*
CBGET	*
CHAIN	*
CINFD	*
CLOSESEG	*
CONFIGH	*
CONF1	*
DATASET	*
DBALTPRI	*
DBALTSEC	*
DBARRAYD	*
DBASE	*
DBBLOCKD	*
DBDACNTL	*
DBDADD	*
DBDEF	*
DBDEFD	*
DBDIRB	*
DBDIRR	*
DBDMPHDR	*
DBEND	*
DBGBLPAK	
BGNWHILE	*
DBITEMD	*
DBLOGCB	*
DBLOGHDR	*
DBPBT	*
DBWAREA	*
DDSBLDL	*
DDSCLOSE	*
GEN370	*
GEN370CK	*
GEN3702	*
GEN73A	*
GEN73LE	*
GEN73NG	
GEN73Z	*
GETARRAY	*
GETBLOCK	*
GETITEM	*
GETLOG	*
GETWA	*
GLOBAL	
GLOBALI	
DDSDCB	*
DDSFIND	*
DDSOPEN	*
DDSSTOW	*
DECDEC	*
DECHEX	*
DEFLOCK	*
DEFMSG	*
DO	*
DOMBOOTH	*
DOMICEXT	*
DOMIRBT	*
DOMIRCME	*
DOMIRCMN	*
DOMIRPRB	*
DOMIRSIO	
DOMISVC1	*
DOMISVC2	*
DOMISVC4	*
DOMSVCN	*
DPACHDEF	
DPATCH	*
DPINIT1	
DPINIT2	
DPINIT3	
DPINIT4	
DPINIT5	
DPLOGDEF	
DPPERMAC	*
DPPFIX	*
DPPFIX2	*
DPPFREE	*
DPPPINIT	*
DPPLEVEL	
DPPSUB	*
DPPSVC	*
DPPSVC9	*
DPPTDSVC	
DPPTETXM	*
ORSELSE	*
PARM	*
PARMDEF	*
PATCH	*
PATCHDEF	
PLISUB	*
PNCHASM	*
PNCHDD	*
PNCHLE	*
PRN	*
PTIME	*
PTIMEDEF	
PTIMEL	*
PTIMRDEF	
DPPTNOTE	*
DPPTPSVC	
DPPTRSVC	
DPPTSETB	*
DPPTWQDL	
DPPXBLKS	*
DPPXBRT	*
DPTDEBUG	
DPTDSVC1	
DPTECBCC	
DPTPSVC1	
DPTPSVC2	
DPTPSVC3	
DPTPSVC4	
DPTPSVC5	
DRECBLKS	*
DUMPLOG	*
DUPDISK	*
ELSE	*
ENDDO	*
ENDIF	*
ENDLOOP	*
ENDSEG	*
ENDSRCH	*
ENTER	*
RQUATE	*
ERRENTER	*
ERRETURN	*
ERREXIT	*
ERRMSG	*
EXIT	*
EXITIF	*
FAILRST	*
FORSUB	*
FREEWA	*
GENCOMCK	
GENEMS	*
GENINIT	*
GEN30	*

GRETURN	*	PTLOGDEF	
GTLOGDEF		PURGEWQ	*
HEADC	*	PUTARRAY	*
HEXDEC	*	PUTBLOCK	*
IF	*	PUTITEM	*
IMP	*	PUTLOG	*
IMPBLKS	*	RECORD	*
ITEM	*	RECRDDEF	
ITEMDEF		REPATCH	*
LENGTH	*	REPCHDEF	
LOCK	*	SETPSW	*
LOCKCBLK	*	STRTSRCH	*
LOG	*	SUPL	*
LOG2	*	SYSLEVEL	
MAINBLOK	*	TIMED	*
MATH	*	TMBIT	*
MESAGDEF		UNTIL	*
MESSAGE	*	VS	*
MSGBLKS	*	WAITDEF	
MSGDEF	*	WHILE	*
MSGEND	*	WTFAILDS	*
MSGRC	*	XIBIT	*
NIBIT	*		
OIBIT	*		
OPENSEG	*		

Basic Object Programs

- DOMIRCPY
- DOMIRFLV
- DOMIRFL2
- DOMIRINT
- DOMIRNIP
- DOMIRWT
- DOMXLIST
- DOMXSTG1
- DPPCALCF
- DPPCPTIM
- DPPCPTIME
- DPPCTIM2
- DPPCTSVC
- DPPCUPCF
- DPPDARAY
- DPPDBLOK
- DPPDBSIF
- DPPDFREQ
- DPPDGETL
- DPPDITEM
- DPPDPUTL
- DPPDRIFE
- DPPDRIFT
- DPPDSUB2
- DPPDUMPL
- DPPDUPDL
- DPPDWRST
- DPPFAONC
- DPPFIXFR
- DPPIDBAS
- DPPIIRB
- DPPILOGN
- DPPINIT0
- DPPINIT1
- DPPIPFIX
- DPPIPFRE
- DPPISTAE
- DPPITIMI
- DPPMINIT
- DPPMMSG
- DPPMMSGV
- DPPMMSG1
- DPPPARM
- DPPPIF
- DPPSAMP1
- DPPSASOC
- DPPSBFST
- DPPSBF1
- DPPSCHCK
- DPPSCHK2
- DPPSCHK3
- DPPSCHK4
- DPPSCHPR
- DPPSCLUP
- DPPSCL1
- DPPSCMPR
- DPPSCP2B
- DPPSCRBK
- DPPSCT2T
- DPPSDDSX
- DPPSDSCB
- DPPSINIT
- DPPSINI2
- DPPSINI3
- DPPSINI4
- DPPSINI5
- DPPSINI6
- DPPSLOCK
- DPPSMSGI
- DPPSMSGO
- DPPSNOTE
- DPPSNTPT
- DPPSOPCL
- DPPSOP1
- DPPSOP2
- DPPSPNTF
- DPPSRCIO
- DPPSRDWT
- DPPSRDW2
- DPPSRLSE
- DPPSRSRV
- DPPSRSTR
- DPPSSHAR
- DPPSSRCH
- DPPSST1
- DPPSSWCH
- DPPSTBOS
- DPPSTKCK
- DPPSUNLK
- DPPSUNSH
- DPPSWRST
- DPPSXTCB
- DPPTCBGT
- DPPTCSVC
- DPPTDLMP
- DPPTETXR
- DPPTGWFW
- DPPTIMPS
- DPPTPMON
- DPPTPWQE
- DPPTQIMP
- DPPTRGWA
- DPPTSMON
- DPPTSTAE
- DPPTWAIT
- DPPTWSVC
- DPPUMSG
- DPPXDBAS
- DPPXDBAT
- DPPXDBCP
- DPPXDBDA
- DPPXDBLG
- DPPXDBLG
- DPPXDBPT
- DPPXDEFL
- DPPXDPB
- DPPXDRC
- DPPXDRCX
- DPPXIMPP
- DPPXIMPW
- DPPXKILL
- DPPXLOCK
- DPPXNRTI
- DPPXPCON
- DPPXRDR
- DPPXRINT
- DPPXRPRT
- DPPXSVCP
- DPPXS2SC
- DPPXUTIL
- DPPZSAMP

Basic Message Programs

DDSMSG
ETXRMSG
FAILRBT
FAILRPRB
FAILRSTM
INITMDCS
SRTOSMSG

Optional Source Programs/Macros

BIGMOVE *
CVDEBC *
DBARBLDL *
DBIITEMR *
DBREFRSH *
DBSORT *
DDSDSECT
DDSNTPT
DDSSULU *
DDSTSSC
DOMIRCPY
DOMIRFLV
DOMIRFL2
DOMIRINT
DOMIRNIP
DOMIRWT
DOMXLIST
DOMXSTG1
DPCALCF1
DPCTIME1
DPCTIME2
DPCTIM21
DPCTIM22
DPCTSVC1
DPCTSVC2
DPCTSVC3
DPCTSVC4
DPCUPCF1
DPCUPCF2
DPCUPCF3
DPCUPCF4
DPIDBAS1
DPIDBAS2
DPIDBAS3
DPIDBAS5
DPIDBAS6
DPINIT01
DPINIT02
DPINIT03
DPINIT04
DPINIT05
DPINIT06
DPINIT07
DPPSSWCH
DPPSTBOS
DPPSTKCK
DPPSUNLK
DPPSUNSH
DPPSWRST
DPPSXTCB
DPPTCBGT
DPPTCSVC
DPPTDLMP
DPPTETXR
DPPTGWFW
DPPTIMPS
DPPTPMON
DPPTPWQE
DPPTQIMP
DPPTRGWA
DPPTSMON
DPPTSTAE
DPPTWAIT

DPINIT08
DPINIT09
DPPINIT0A
DPINIT11
DPINIT12
DPINIT13
DPINIT14
DPITIMI1
DPPCALCF
DPPCPTIM
DPPCTIME
DPPCTIM2
DPPCTSVC
DPPCUPCF
DPPDARAY *
DPPDASUB
DPPDBLOK
DPPDBSIF
DPPDFREQ
DPPDGETL
DPPDITEM
DPPDPUTL
DPPDRIFE
DPPDRIFT *
DPPDSTRT
DPPDSUB2
DPPDUMPL
DPPDUMPL
DPPDWRST
DPPFAONC
DPPFIXFR
DPPIDBAS
DPPIIRB
DPPILOGN
DPPINIT0
DPPINIT1
DPPIPFIX
DPPIPFRE
DPPISTAE
DPPITIMI
DPPMINIT
DPPMMSG
DPPMMSGV
DPPMMSG1
DPPXSVCP
DPPXS2SC
DPPXUTIL
DPPZSAMP
DPTCSVC1
DPTDLMP1
DPTDLMP2
DPTDLMP3
DPTDLMP4
DPTDLMP5
DPTPMON1
DPTPMON2
DPTPMON3
DPTPMON4
DPTPMON5
DPTSMON1
DPTWSVC1
DPTWSVC2
DPTWSVC3

DPPPARM
DPPPIF
DPPSAMP1
DPPSASOC
DPPSBFST
DPPSBF1
DPPSCHCK
DPPSCHK2
DPPSCHK
DPPSCHK4
DPPSCHPR
DPPSCLUP
DPPSCL1
DPPSCMPR
DPPSCP2B
DPPSCRBK
DPPSCT2T
DPPSDDSX
DPPSDSCB
DPPSINIT
DPPSINI2
DPPSINI3
DPPSINI4
DPPSINI5
DPPSINI6
DPPSLOCK
DPPSMSGI
DPPSMSGO
DPPSNOTE
DPPSNTPT
DPPSOPCL
DPPSOP1
DPPSOP2
DPPSPNTF
DPPSRCIO
DPPSRDWT
DPPSRDW2
DPPSRLSE
DPPSRSRV
DPPSRSTR
DPPSSHAR
DPPSSRCH
DPPSST1
PSECTEND *
PWQE *
QHBK *
QPBK *
RCALL *
RCSHEAD *
RLMHEAD *
SETO *
SETPM *
STAEBLK *
STAEXBK *

DPPTWSVC
DPPUMSG
DPPUMSG1
DPPUMSG2
DPPXDBAS
DPPXDBAT
DPPXDBCP
DPPXDBDA
DPPXDBIN
DPPXDBLG
DPPXDBPT
DPPXDEFL
DPPXDPB
DPPXDRC
DPPXDRCX
DPPXIMPP
DPPXIMPW
DPPXKILL
DPPXLOCK
DPPXNRTI
DPPXPCON
DPPXRDR
DPPXRINT
DPPXRPRT
DPXDBIN1
DPXDBIN2
DPXDBIN3
DPXDBIN4
DPXDBIN5
DPXDBIN6
FINDPARM *
GMSG *
HEXEBC *
INITCB *
IPROB *
LTYP *
MARKMASK *
MASKDATA *
MXSTG101
MXSTG102
MXSTG103
MXSTG104
MXSTG105
PPLPTPCH
PPLSCAN
PPLSCAN2
PPLUPDTE
PSECT *

APPENDIX G. GLOSSARY

This manual introduces many new terms and acronyms not commonly used in data processing. This glossary defines those terms unique to or having a special meaning in this program and this manual. Accordingly, terms which are included in the IBM Data Processing Glossary (GC20-1699) are not included here.

Term — Definition

Array — An arrangement of data items in one or more dimensions.

Block — One or more data items. One or more blocks of equal dimensions make up an array.

Blocked — An array consisting of one or several blocks. A blocked array may be either VS or DA resident and may be accessed through GETBLOCK and PUTBLOCK.

Backup — The secondary copy of a duplicate data set pair.

Continuous — A program that resides and executes in the monitor online CPU and monitors specified storage locations to ensure that the online system is functioning.

DA Resident — A term used to group arrays by their residence during online operation. DA resident specifies that the array resides on direct access storage during online operation.

Data Base — The collection of data arrays and control information consisting of one partitioned data set, and one or more direct data sets. During real-time processing, part of data base will be loaded into virtual storage.

Dependent Task — A task created by the Special Real Time Operating System without a task name. A dependent task executes only once.

Duplicate Data Set — A feature of the Special Real Time Operating System that allows for duplicate copies of critical data sets to be maintained by executing duplicate I/O accesses.

Failover — The procedure by which the backup CPU is made to become the primary CPU.

Independent Task — A task created by the Special Real Time Operating System with a task name. An independent task remains in existence after it has executed and is capable of executing program cyclicly.

Item — One member of a group, one or more items make up an array.

Lock — A method of controlling a resource so that only one program may use the resource at a time.

Log (Logging) — The process of copying a VS resident array to a log dataset (DA resident).

Log Array A DA resident array that will contain the copy or copies of the VS resident loggable array.

Log Header The control information associated with the VS resident loggable array.

Master The controlling partition in a two partition operation.

Normal Start The process by which the Special Real Time Operating System is initializing from the control statements in the input stream using the initial data loggable arrays.

Offline That processing which is executed not under control of the Special Real Time Operating System, i.e., processing in another CPU or in a non-real time partition.

Online That processing which is executed under control of the Special Real Time Operating System, i.e., as a subtask of the Special Real Time job step task itself.

Playback The process by which the data previously collected by the data record routine is retrieved and deformatted.

Primary The main CPU in a multiple CPU environment; i.e., the CPU which is currently controlling the functions for the real-time environment. Also the main copy of a duplicate data set group.

PROBE That program that runs in the backup computer and tests the continuous monitor in the online CPU. If the value on the direct control static data lines fails to change during twice of the update interval, the PROBE recommends a failure.

Record The processing of collecting specified data and saving it in a formatted mode for later retrieval by the playback function.

Refresh The process by which the Special Real Time Operating System is initialized using the most recent copies of loggable arrays. A refresh start may be from the input stream or from a failure data set.

Resource Table An 8-byte area provided by the Special Real Time Operating System for each real-time task.

Restart The procedure by which a real-time CPU is reinitialized to the point at which the failover restart data set was written.

SLAVE That partition in a two-partition real-time operating system which contains only some of the Special Real Time Operating System services. The services not contained in the SLAVE partition are provided by the MASTER partition.

Status Panel A user fabricated hardware panel used to indicate which CPU is primary and which is backup in a real-time environment and also to indicate when and how a failover is to occur.

Time Drift The variation of the System/370 Time of Day Clock from the absolute time.

Unblock The freeing of a resource that had previously been reserved by the LOCK function.

Unblocked The freeing of a resource that had previously been reserved by the LOCK function.

VS Resident An array that resides in virtual storage.

IBM

International Business Machines Corporation

General Systems Division
5775D Glenridge Drive N.E.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza
New York, New York 10017
(International)

Special Real Time Operating System Programming RPQ Description and Operation Manual Printed in USA SH20-1773-0