IBM

Series/1

GC34-0102-0

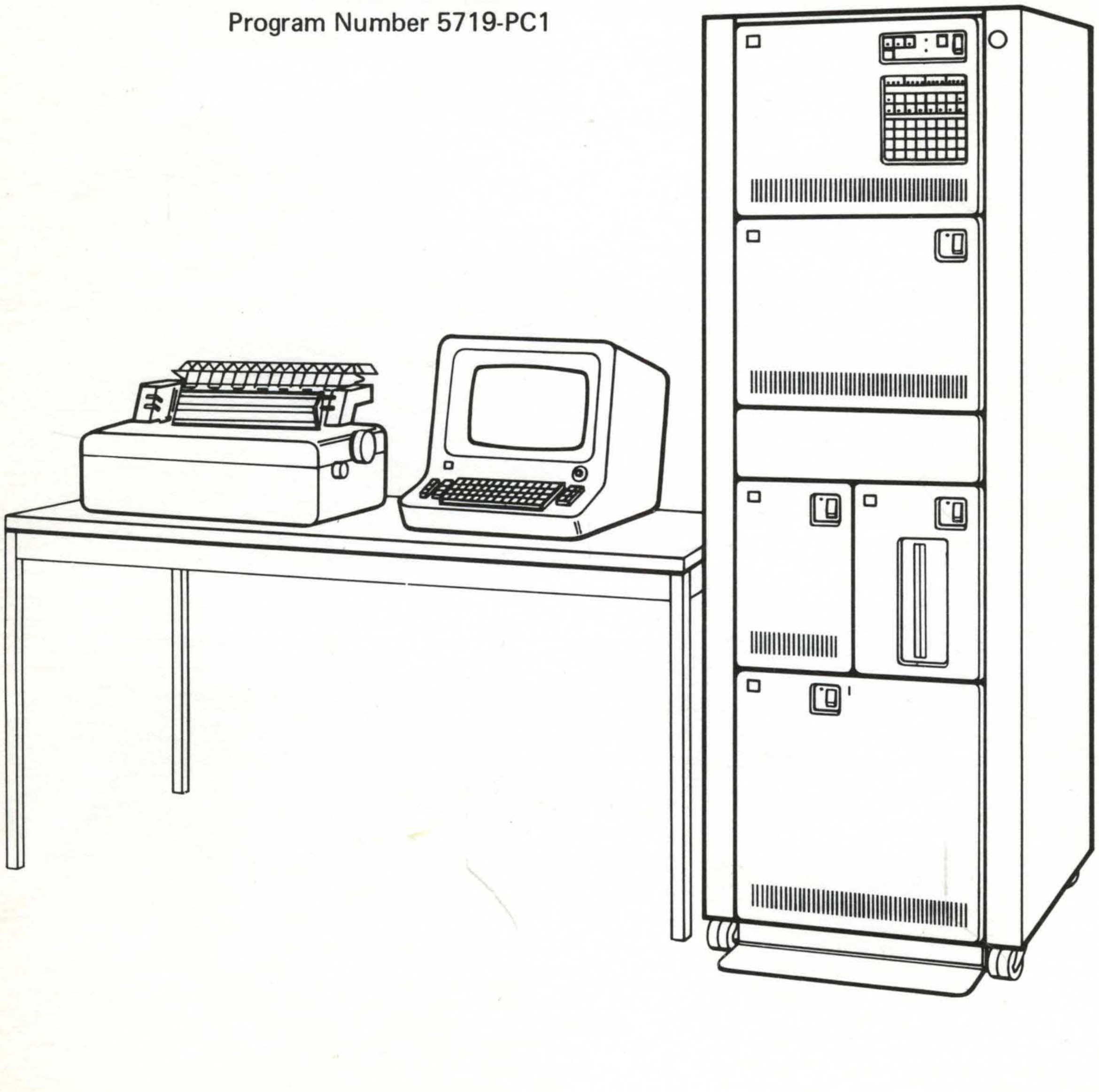
S1-20

PROGRAM
PRODUCT

INTRODUCTION/PLANNING GUIDE

IBM Series/1 Realtime Programming System Introduction and Planning Guide

Program Number 5719-PC1



IBM

Series/1

GC34-0102-0

S1-20

PROGRAM
PRODUCT

INTRODUCTION/PLANNING GUIDE

IBM Series/1 Realtime Programming System Introduction and Planning Guide

Program Number 5719-PC1

First Edition (February 1977)

This manual applies to the IBM Series/1 Realtime Programming System, Program Number 5719-PC1.

Significant changes or additions to the contents of this publication will be reported in subsequent revisions or Technical Newsletters. Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, send your comments to IBM Corporation, Systems Publications, Department 27T, P.O. Box 1328, Boca Raton, Florida 33432. Comments become the property of IBM.

Contents

Preface

The purpose of this book is to give you an introduction to the software that is available for the Series/1 and, specifically, to introduce you to the features of the Realtime Programming System. This book is also intended to give you the information you need to plan your Series/1 software system. The reader of this book is assumed to be an experienced programmer who is familiar with realtime programming.

How this Book is Organized

Chapter 1. The Total System. This chapter gives an overview of the Series/1 software offering. It briefly describes the program products and how they fit together to form a total system.

Chapter 2. The Operating System: Supervisor Services. This chapter describes the supervisor services—the part of the operating system that helps you organize your programs into an efficient, smooth-running application.

Chapter 3. The Operating System: Data Management. This chapter describes the data-management services of the operating system, which handle the data sets and devices used by your application.

Chapter 4. The Operating System: Communications. This chapter discusses the part of the operating system that directs the transfer of data between your programs and remote stations.

Chapter 5. The Operating System: Utilities. This chapter lists the stand-alone and system utilities—a part of the operating system with which you can easily and efficiently manage data and maintain your system.

Chapter 6. Program Preparation. This chapter describes the program-preparation facilities and high-level languages with which you code and prepare the application programs to be run under the operating system.

Appendixes. The Appendixes give (1) a matrix of system functions and the ways of using them, (2) a glossary of terms, (3) a list of debugging aids, and (4) a list of the options available during system generation.

Related Publications

These publications, when available, will give more detailed information about the topics mentioned in this book:

Programming Publications

- *IBM Series/1 Realtime Programming System: Macro User's Guide—Supervisor*
- *IBM Series/1 Realtime Programming System: Macro User's Guide—Data Management*
- *IBM Series/1 Realtime Programming System: Macro User's Guide—Communications*
- *IBM Series/1 Realtime Programming System: Operator Commands and Utilities*

- *IBM Series/1 Realtime Programming System: Generation and Installation Procedures*
- *IBM Series/1 Realtime Programming System: Macro Reference*
- *IBM Series/1 Realtime Programming System: Messages and Codes*
- *IBM Series/1 Realtime Programming System: Control Blocks and Debugging Guide*
- *IBM Series/1 Program Preparation Subsystem: Introduction*, GC34-0121*
- *IBM Series/1 Program Preparation Subsystem: Batch User's Guide*
- *IBM Series/1 Program Preparation Subsystem: Text Editor User's Guide*
- *IBM Series/1 Program Preparation Subsystem: Macro Assembler User's Guide*, SC34-0124*
- *IBM Series/1 Program Preparation Subsystem: Macro Assembler Reference Summary*
- *IBM Series/1 Program Preparation Subsystem: Application Builder User's Guide*
- *IBM Series/1 Program Preparation Subsystem: Messages and Codes*
- *IBM Series/1 FORTRAN IV: Introduction*, GC34-0132*
- *IBM Series/1 FORTRAN IV: Language Reference*, GC34-0133*
- *IBM Series/1 FORTRAN IV: User's Guide*
- *IBM Series/1 FORTRAN IV: Language Reference Card*
- *IBM Series/1 Mathematical and Functional Subroutine Library: Introduction*, GC34-0138*
- *IBM Series/1 Mathematical and Functional Subroutine Library: User's Guide*
- *IBM Series/1 PL/I: Introduction*, GC34-0084*
- *IBM Series/1 PL/I: Language Reference Manual*
- *IBM Series/1 PL/I: User's Guide*
- *IBM Series/1 PL/I: Messages*
- *IBM Series/1 PL/I: Installation Guide*

*These publications are available now.

Hardware Publications

The following hardware publications are all available now:

- *IBM Series/1 Model 5 4955 Processor and Processor Features Description*, GA34-0021
- *IBM Series/1 Model 3 4953 Processor and Processor Features Description*, GA34-0022
- *IBM Series/1 4962 Disk Storage Unit Description and 4964 Diskette Unit Description*, GA34-0024
- *IBM Series/1 4973 Printer Description*, GA34-0044
- *IBM Series/1 4974 Printer Description*, GA34-0025
- *IBM Series/1 4979 Display Station Description*, GA34-0026
- *IBM Series/1 System Summary*, GA34-0035-1

Chapter 1. The Total System

The total software system for the Series/1 is made up of these program products:

- The IBM Series/1 Realtime Programming System (also referred to as *the operating system*), which comprises supervisor services, data management, communications, and utilities
- The IBM Series/1 Program Preparation Subsystem, which comprises the tools needed to prepare the programs that run under the operating system
- IBM Series/1 FORTRAN IV
- The IBM Series/1 Mathematical and Functional Subroutine Library
- IBM Series/1 PL/I

The Operating System

The operating system is the basic control program upon which applications are built. It is flexible and is suitable for a wide variety of applications. The operating system controls and manages system resources—processor storage and devices. It is a *multiprogramming, multitasking, event-driven, disk-based* system that is the environment for both realtime and batch applications:

- *Multiprogramming* — Processor storage is divided into multiple fixed *partitions*; programs execute in partitions based on their priority. Programs in two or more partitions are processed concurrently.
- *Multitasking* — The operating system allows multiple concurrent task operations in the same partitions with synchronization and communication between them. In addition, single reenterable programs can be used by more than one task.
- *Event-driven* — Programs are queued for execution in partitions based on these types of events:
  - External (process interrupt)
  - Time of day
  - Time interval, either single or repetitive
  - Operator request
  - Program request
- *Disk-based*—The system program library and transients must reside on disk. Your program libraries and application program overlays can reside on disk or diskettes.

This chapter discusses the overall function of the operating system—its individual features are discussed in more detail in Chapters 2, 3, 4, and 5.

Hardware Requirements and Options

The minimum hardware required to install the operating system is:

- Processor—IBM 4953 Processor or IBM 4955 Processor, with at least 48KB of processor storage. If you use the Series/1 PL/I compiler with the operating system, your processor must have at least 64KB of processor storage.
- Operator Station—either:
  - An IBM 4979 Display Station
  - A teletypewriter or other device that is compatible with the Teletypewriter Adapter Feature. If this type of operator station is chosen, the processor must be equipped with the Teletypewriter Adapter Feature (#7850).

- IPL devices—one IBM 4962 Model 1 or 1F Disk Storage Unit *and* IBM 4964 Diskette Unit OR one IBM 4962 Model 2 or 2F Disk Storage Unit (combination disk/diskette unit).
- Hard-copy device—either:
  - The IBM 4973 Line Printer
  - The IBM 4974 Printer
  - The teletypewriter or other device that is compatible with the Teletypewriter Adapter Feature. If this type of hard-copy device is chosen, the processor must be equipped with the Teletypewriter Adapter Feature (#7850).

  Note. The operating system permits more than one of all the devices mentioned in the preceding list, with the exception of the processor.

The optional hardware that is available for use with the operating system is:

- The IBM 4982 Sensor Input/Output Unit, which supports:
  - Analog input
  - Analog output
  - Digital input
  - Digital output
- Integrated Digital Input/Output Non-Isolated (#1560).
- Timers (#7840).
- Communications features for asynchronous (start-stop) and binary synchronous communications:
  - Asynchronous Communications Single Line Control (#1610)
  - Binary Synchronous Communications Single Line Control (#2074)
  - Binary Synchronous Communications Single Line Control—High Speed (#2075)
  - Asynchronous Communications 8 Line Control (#2091)
  - Asynchronous Communications 4 Line Adapter (#2092)
  - Binary Synchronous Communications 8 Line Control (#2093)
  - Binary Synchronous Communications 4 Line Adapter (#2094)
  - Communications Indicator Panel (#2000)

  Note. Communications supports two types of terminals—the IBM 2740 Model 1 Communications Terminal, and the Teletype Model 33/35 (trademark of the Teletype Corporation) or equivalent.

  The operating system also supports binary synchronous communications with an IBM System/370 using OS/VS1 BTAM in point-to-point connections.
- Floating-point (#3920).

  Note. If your system is not equipped with this feature, but you wish to perform floating-point operations, the operating system has a floating-point emulator program that you can specify during system generation.
- Programmer console (#5650).
- Battery Backup Unit (#4999).

Partitions, Task Sets, and Tasks

Processor storage is divided into *partitions*. There can be as few as 2 partitions, or as many as 16. The supervisor occupies partition 0.

Applications reside in partitions as *task sets*, and only one task set can occupy a partition at a time. The operating system *does* permit exchanging the task set that is resident in a partition for another task set.

Within a task set there can be one or more units of code called *programs*. A *task* represents a single thread of execution through one or more programs.

The task is logically the basic execution unit under the operating system, and there can be as many tasks and programs as there is storage to execute them. In addition, tasks can execute either storage-resident or disk-resident programs, and there can be as many disk-resident programs as there is disk space to hold them. Tasks can be activated as a result of:

- Process interrupts
- Requests from the operator station
- Programs
- Timer services
- Other tasks

Before a task set can be loaded into a partition for execution, it must reside in a data set on a disk volume called a *task set library*. An application program can use one or more I/O devices or data sets. These devices and data sets are described symbolically within the program, so that you can specify them externally from the program. These specifications must be processed and stored in the task set library before the program refers to the devices or data sets. For improved performance of your application programs, the operating system has a method by which you can *bind* a task set to a partition and to physical devices before the task set is activated.

A more detailed discussion of tasks and task sets is in Chapter 2.

Operator Interaction

The operator can control and interact with the operating system while it is running. The operating system does not require that an operator be present for the system to operate; however, the application programmer can, if he wishes, require an operator response to certain requests. Here is a partial list of things an operator can do with operator commands:

- Cause execution of a task set
- Cancel a task set within a partition
- Reply to requests from programs
- Set devices online and offline
- Alter assignment of device numbers

Input/Output

The operating system has three levels of I/O interface for application programs:

- Basic (EXIO)—access by physical record, with a full range of device control
- Physical (READ/WRITE)—access by block; control is returned to the next sequential instruction of the application program while the I/O operation is taking place
- Logical (GET/PUT)—access by logical record; you can optionally request that control be returned to the next sequential instruction of the application program while the I/O operation is taking place

The level of I/O interface used depends on the needs of the particular application program.

For these data processing devices, you can use either a READ/WRITE, GET/PUT or EXIO interface:

- Disks
- Diskettes
- Printers
- Teletypewriters
- Display stations

Timers are handled through a READ/WRITE or EXIO interface.

Sensor I/O is handled through a READ/WRITE or EXIO interface for both analog and digital I/O.

For start-stop and binary synchronous communications devices, you use READ/WRITE.

The operating system permits three types of data set organizations:

- Consecutive—records are processed in a sequential, order-of-arrival basis, and an end-of-data marker is maintained.
- Random—each record is individually addressed, and no end-of-data marker is maintained.
- Partitioned—records are organized as a collection of data sets with a directory that points to each data set or member.

Through message buffering, the application program can send messages to a printer, communications device, or the operator station and have these messages intermediately queued on disk.

Details about data management are discussed in Chapter 3.

Communications

The operating system has the necessary supervisory routines for applications that use communications. Communications support directs the transfer of data between programs and remote stations—a remote station can be either a terminal or another computer. Communications supports switched (dial-up) or non-switched point-to-point connections using binary synchronous or start-stop line control.

For more information about communications, refer to Chapter 4.

Utilities

The operating system has a set of utilities for installation and maintenance of application programs and data. There are *stand-alone* utilities, which are loaded from diskette, and *system* utilities, which reside on disk. The system utilities are invoked at the operator station and can run concurrently with an application program.

For information about individual utility programs, see Chapter 5.

Program Preparation

The program preparation tools are separate program products. With them, you prepare the programs that run under the operating system. Each program preparation facility runs as a task set under the operating system in a batch environment. The program preparation facilities are:

- The Program Preparation Subsystem, which comprises:
  - The job stream processor
  - The text editor
  - The macro assembler
  - The application builder
- FORTRAN
- The Mathematical and Functional Subroutine Library
- PL/I

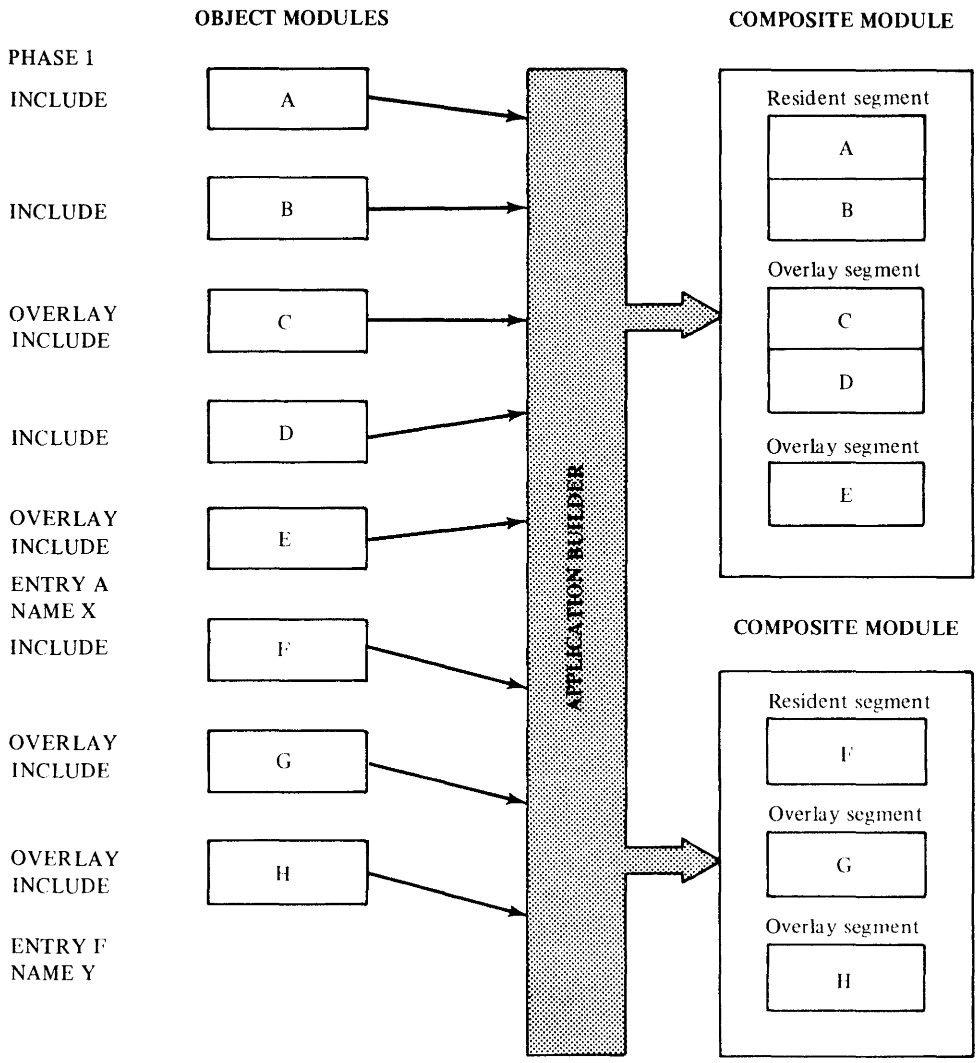
An application goes through several stages before it is ready for execution under the operating system. First, it is coded as a set of programs—individual source modules in either assembler language, FORTRAN, or PL/I. You then enter control statements to direct the preparation process. The source programs are entered into the system through the text editor, then they are assembled or compiled. Next, the application builder combines the individual object modules into a single storage load that is executable under the operating system.

For an explanation of each program preparation facility, see Chapter 6.

How Programs Flow through the Total System

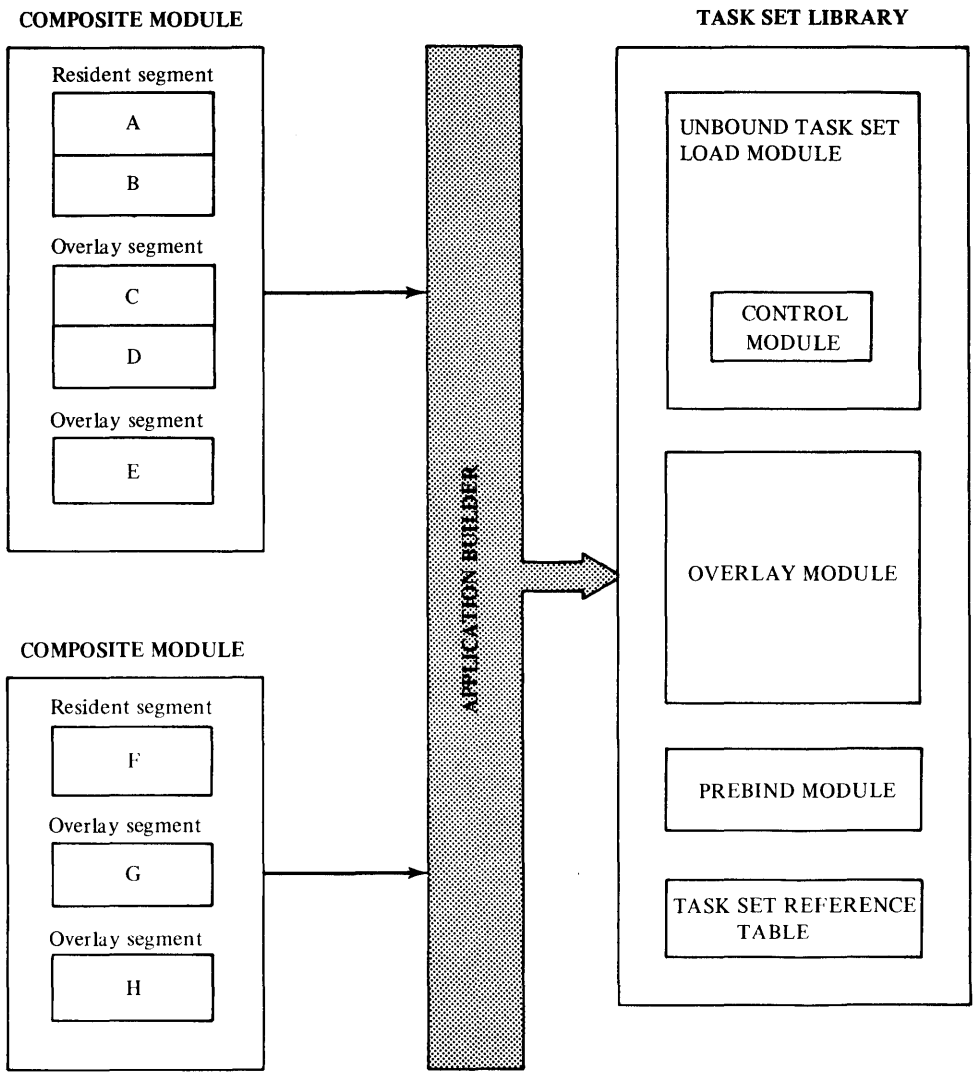
Once your programs have been assembled or compiled, they reside on disk or diskette as *object modules,* and they must be further processed to create a task set.

In your instructions to the application builder, you indicate which object modules are to be grouped together as *composite modules,* and specify the entry point of each composite module. You also specify which programs are to be in *resident segments* and which ones are to be in *overlay segments.*



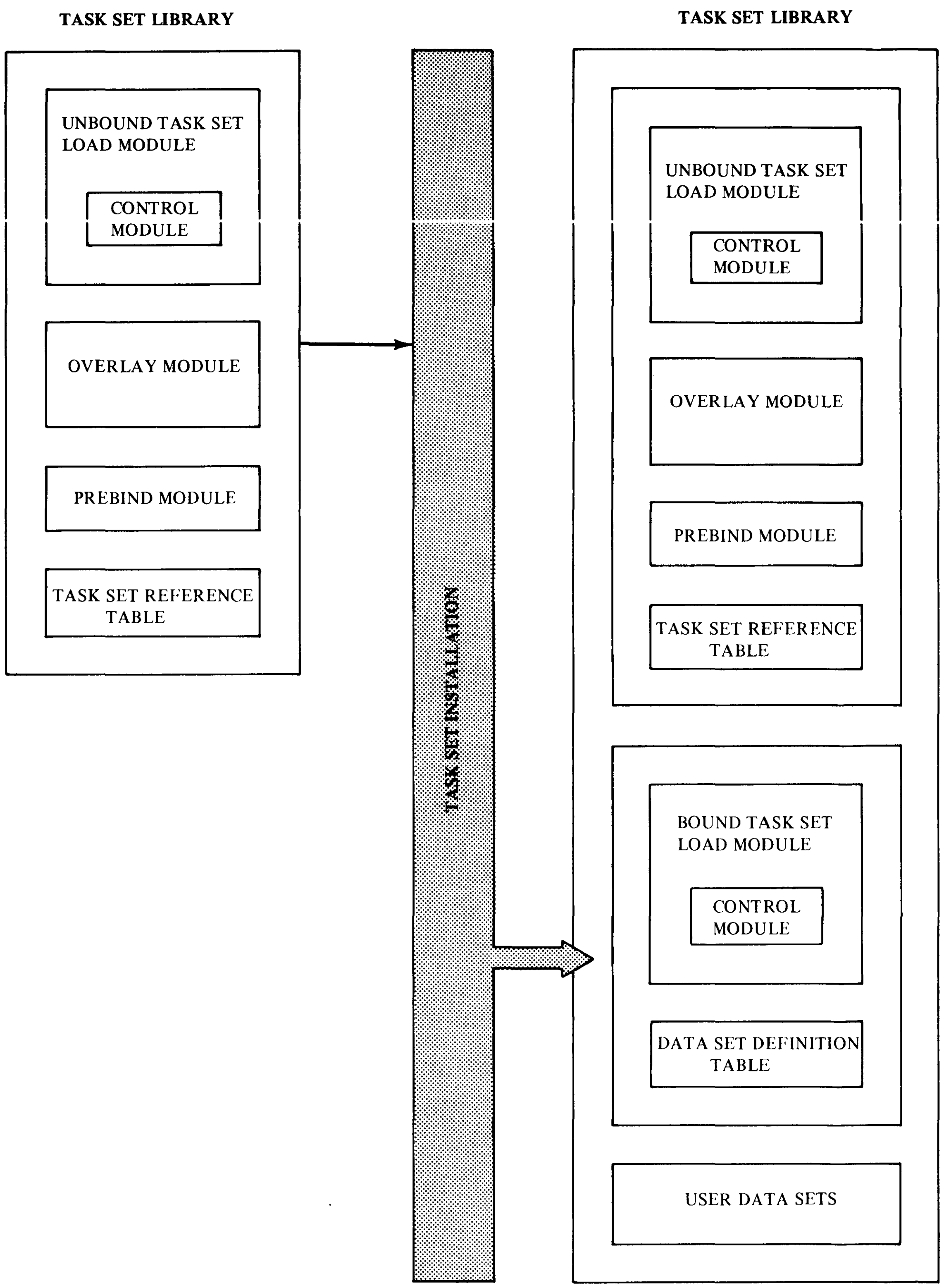
The composite modules are an intermediate output of the application builder. Each composite module consists of a resident segment and any overlay segments that you have defined for that composite module. The names X and Y are the names of the data set members that contain the composite modules.

The *task set library* is the final output of the application builder, and it resides on disk or diskette. To create the task set library, the application builder combines all the resident segments (from the composite modules) into an *unbound task set load module*, which includes a *control module* containing information needed by the operating system. The application builder combines all the overlay segments into an *overlay module*. Finally, the application builder adds a *prebind module* (optional) and a *task set reference table*, which also contain information needed by the operating system.



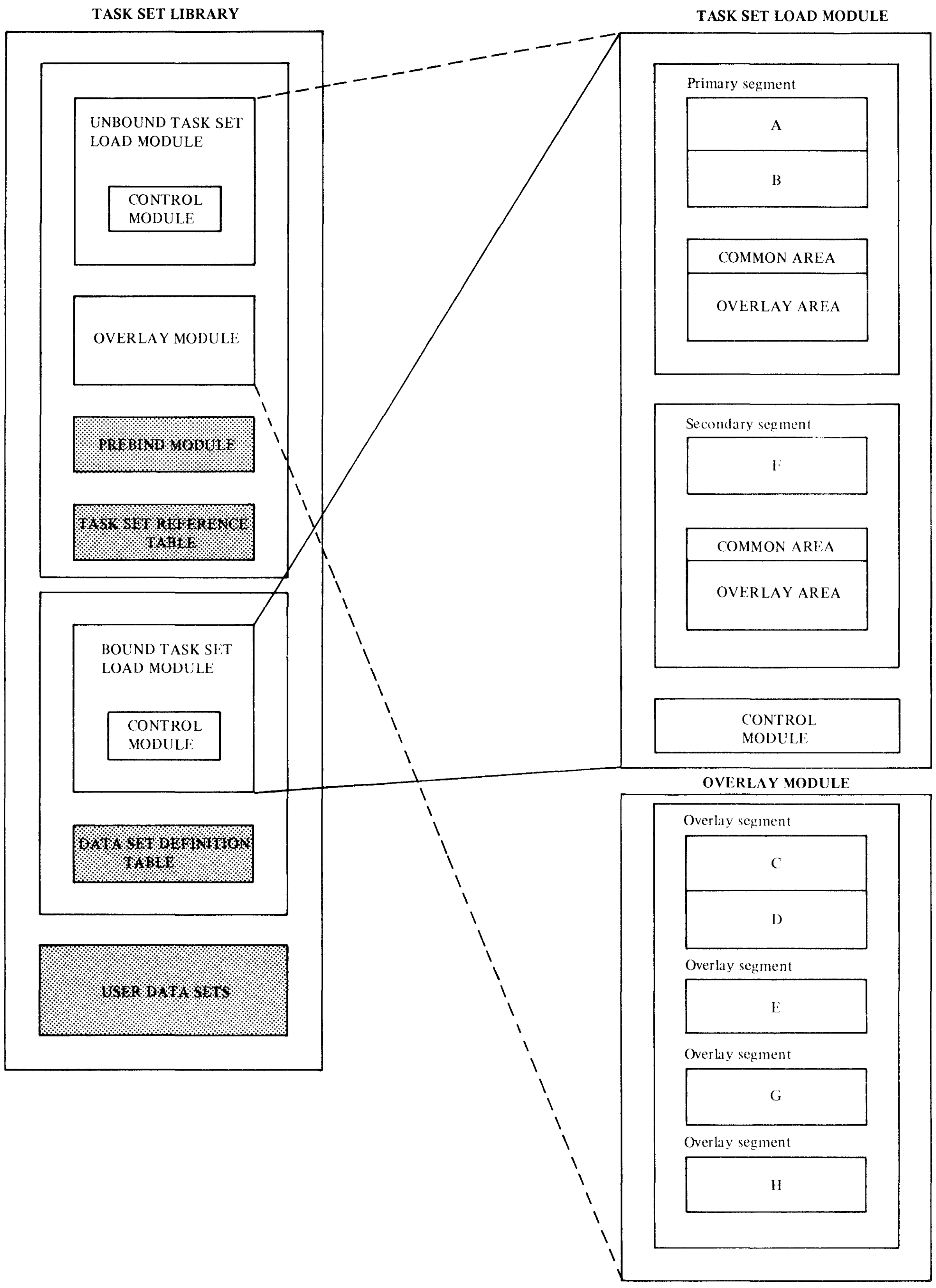
Up to this point you can, if you wish, do program preparation on a system other than the one on which the task set is to execute. If you do that, you can now copy the task set library onto a portable medium, such as a diskette, and restore it on the execution-time system.

At this point in the discussion, it is necessary for you to understand *binding* — the assignment or connection of resources to a task set *before* the task set is executed. Binding can be done early (this is also called *prebinding*) or it can be done late. If you choose to do early binding, you indicate to the application builder which items you want bound. This information is stored in the prebind module associated with the unbound task set load module. The actual binding occurs when the task set is installed, at which time the installation process adds a *bound task set load module* to the task set library. The task set library can also contain a table of data set definitions and other data sets required by the task set.

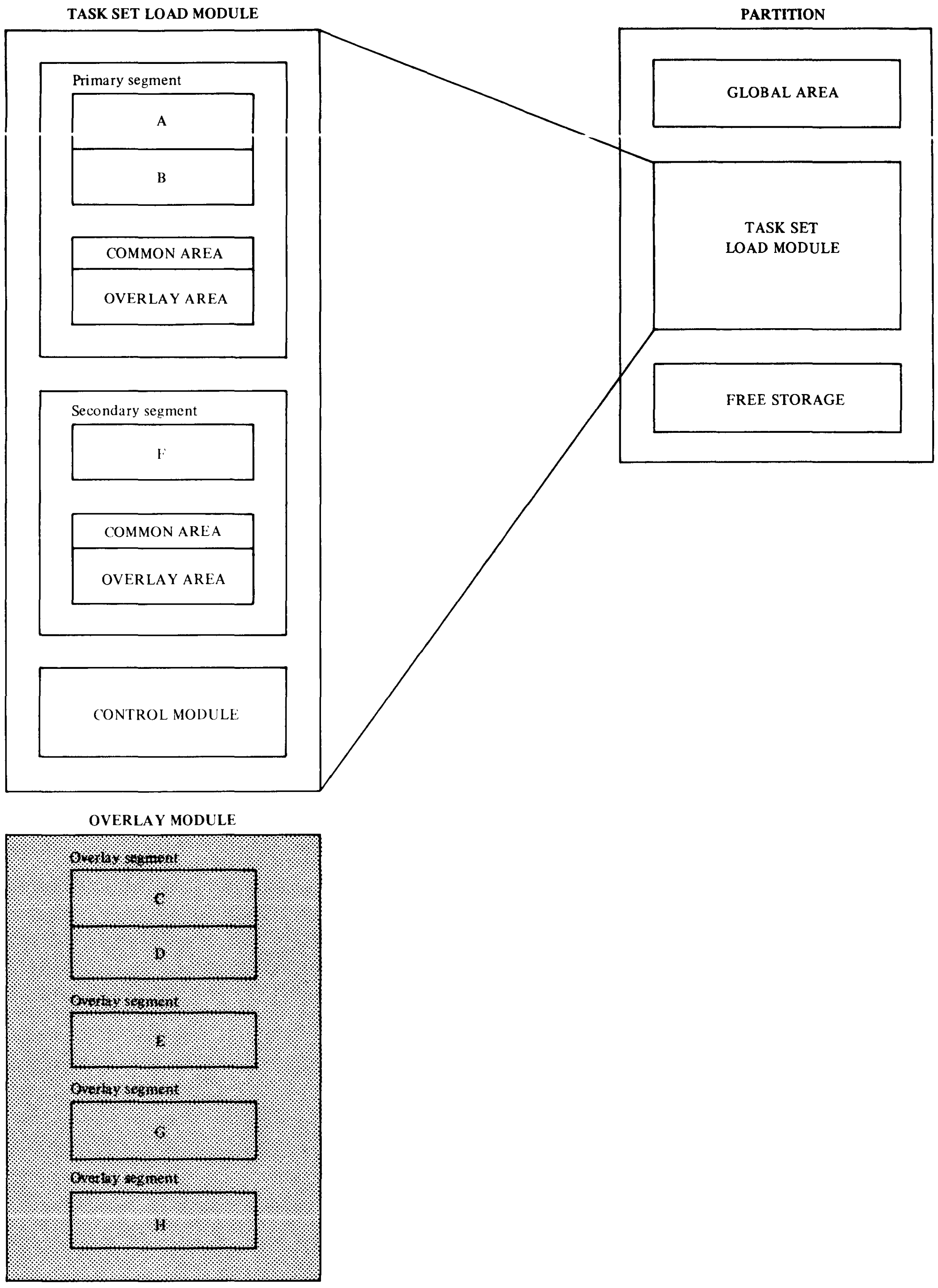


If you choose to do late binding, you need not specify binding information to the application builder, nor do you need to install the task set. At execution time, the operating system must assign resources to the unbound task set. Early binding is, therefore, a way to eliminate the time needed to assign resources during execution of a task set.

The following diagram expands the contents of the task set library. Here you can see where the original object modules are located in an executable task set.

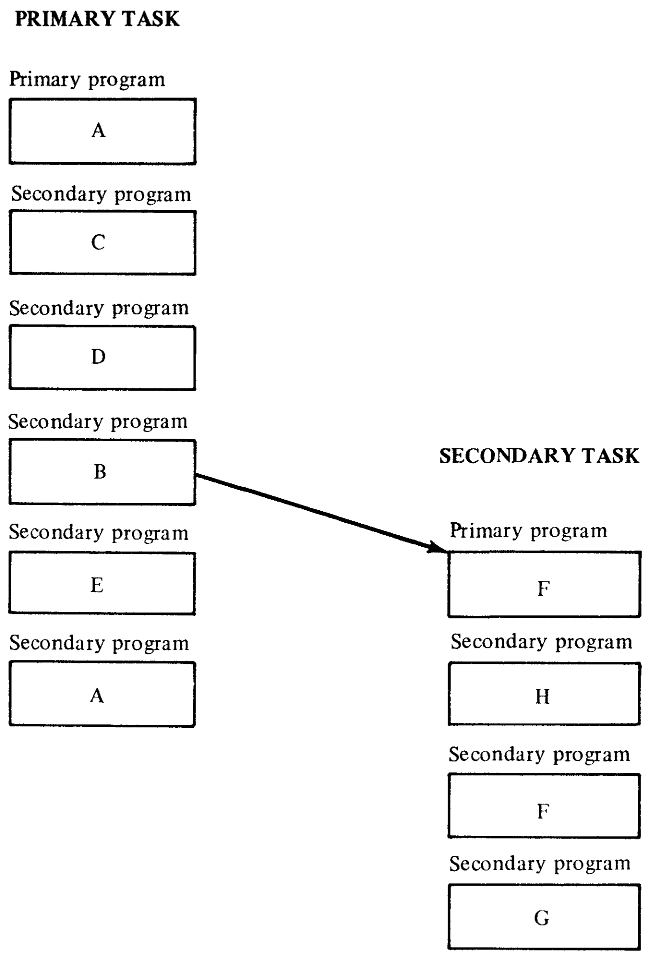


When a task set is ready for execution, the task set load module is placed into the specified partition.



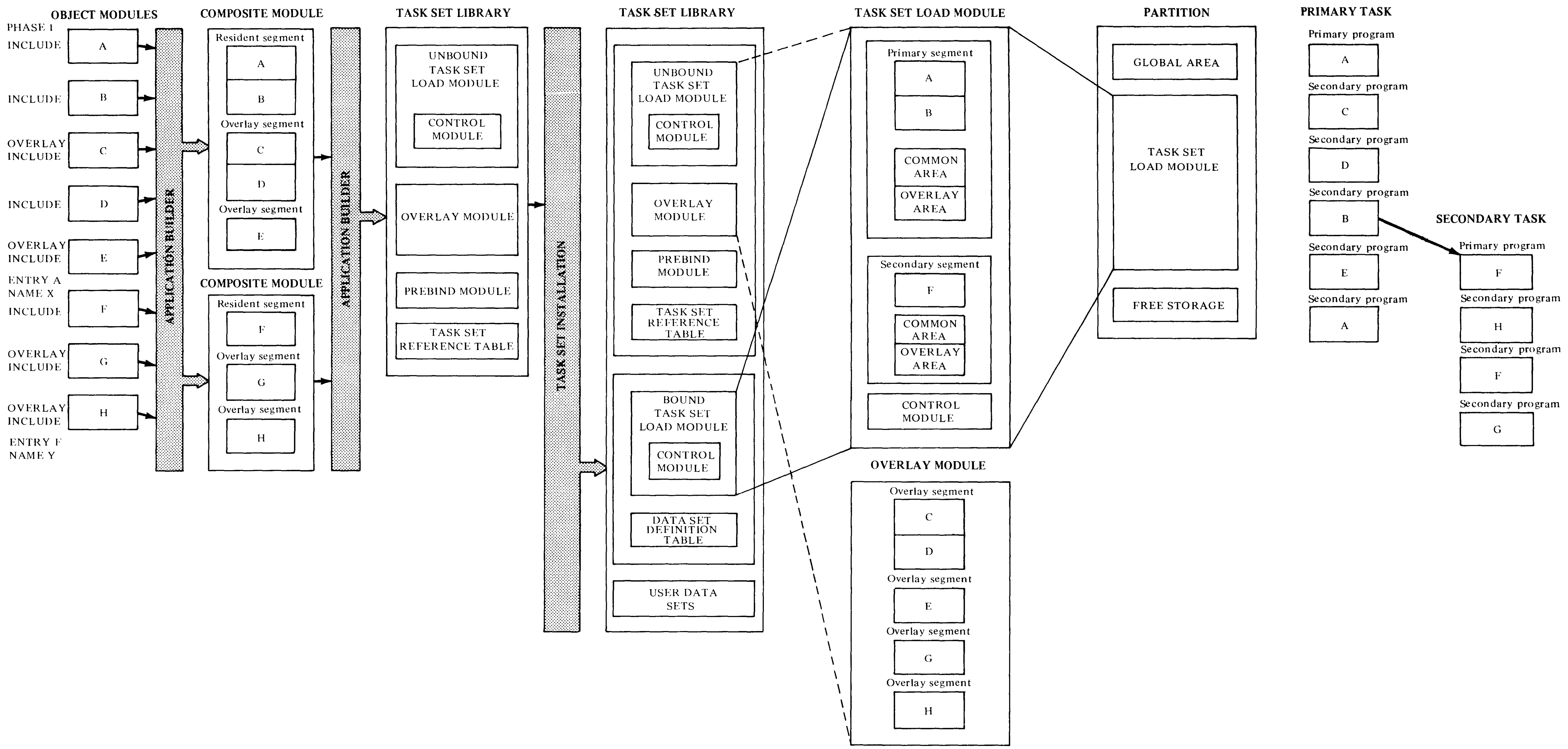
Note that the partition also contains a *global area* (a data area addressable by all programs in the task set) and some *free storage* (storage, within the partition, that is available for use by the task set).

The following illustration shows one of the many possible ways execution can proceed within the task set.



Execution proceeds in *tasks* — threads of execution through programs. The first task to execute when the task set gains control is called the *primary task*. The first program of a task is the *primary program*. The primary program can start other programs, which are called *secondary programs*. Secondary programs can, in turn, start other secondary programs. A *secondary task* can be started by any program in the task set.

The diagram on the following page summarizes the flow of programs through the total system. It is a combination of all the diagrams shown in the previous discussion, and it should help you understand how all the steps fit together.


OBJECT MODULES
PHASE 1
INCLUDE A
INCLUDE B
OVERLAY INCLUDE C
INCLUDE D
OVERLAY INCLUDE E
ENTRY A
NAME X
INCLUDE F
OVERLAY INCLUDE G
OVERLAY INCLUDE H
ENTRY F
NAME Y
APPLICATION BUILDER
COMPOSITE MODULE
Resident segment
A
B
Overlay segment
C
D
Overlay segment
E
COMPOSITE MODULE
Resident segment
F
Overlay segment
G
Overlay segment
H
APPLICATION BUILDER
TASK SET LIBRARY
UNBOUND TASK SET LOAD MODULE
CONTROL MODULE
OVERLAY MODULE
PREBIND MODULE
TASK SET REFERENCE TABLE
TASK SET INSTALLATION
TASK SET LIBRARY
UNBOUND TASK SET LOAD MODULE
CONTROL MODULE
OVERLAY MODULE
PREBIND MODULE
TASK SET REFERENCE TABLE
BOUND TASK SET LOAD MODULE
CONTROL MODULE
DATA SET DEFINITION TABLE
USER DATA SETS
TASK SET LOAD MODULE
Primary segment
A
B
COMMON AREA
OVERLAY AREA
Secondary segment
F
COMMON AREA
OVERLAY AREA
CONTROL MODULE
OVERLAY MODULE
Overlay segment
C
D
Overlay segment
E
Overlay segment
G
Overlay segment
H
PARTITION
GLOBAL AREA
TASK SET LOAD MODULE
FREE STORAGE
PRIMARY TASK
Primary program
A
Secondary program
C
Secondary program
D
Secondary program
B
Secondary program
E
Secondary program
A
SECONDARY TASK
Primary program
F
Secondary program
H
Secondary program
F
Secondary program
G

FOLD THIS PAGE OUT

Chapter 2. The Operating System: Supervisor Services

Put simply, supervisor services control and distribute the resources of the system. This chapter describes the services of the supervisor in a top-down manner. It begins by discussing the management of *processor storage* — the fundamental resource of any system. Next follows a discussion of *partitions* — divisions of processor storage. This chapter then describes *task sets* — the code that executes in partitions, then *tasks* — single threads of execution through one or more programs. *Programs* execute under tasks. Finally, this chapter describes the individual services (such as queuing, events, and timers) that can be used by tasks.

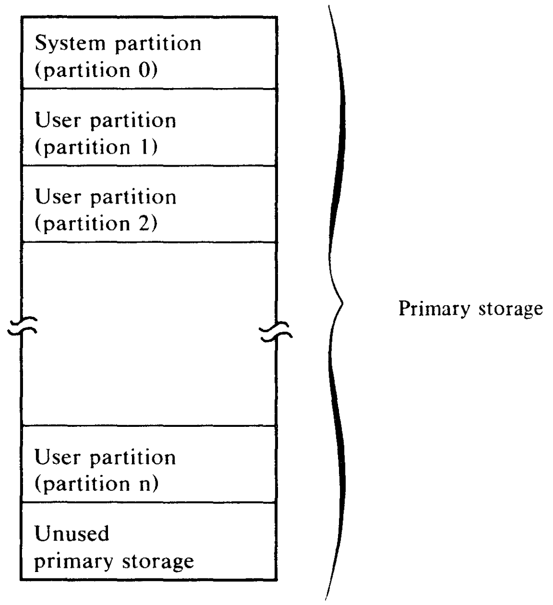
Processor Storage

Storage management controls *primary storage* — storage that is within the first 64KB of processor storage. The supervisor does not support storage (if any) beyond the first 64KB.

Partitions

The supervisor manages storage for task set execution in *partitions*. A partition is a contiguous storage area of fixed size that you specify at system generation. A partition begins and ends on a 2KB boundary, and each partition has a unique numeric identifier (0—15). Any storage not allocated to a partition is called *unused primary storage*.

You can define 2 to 16 partitions. The minimum configuration is the system partition (always partition 0) plus one user partition; the maximum configuration is the system partition plus 15 user partitions.


System partition (partition 0)
User partition (partition 1)
User partition (partition 2)
User partition (partition n)
Unused primary storage
Primary storage

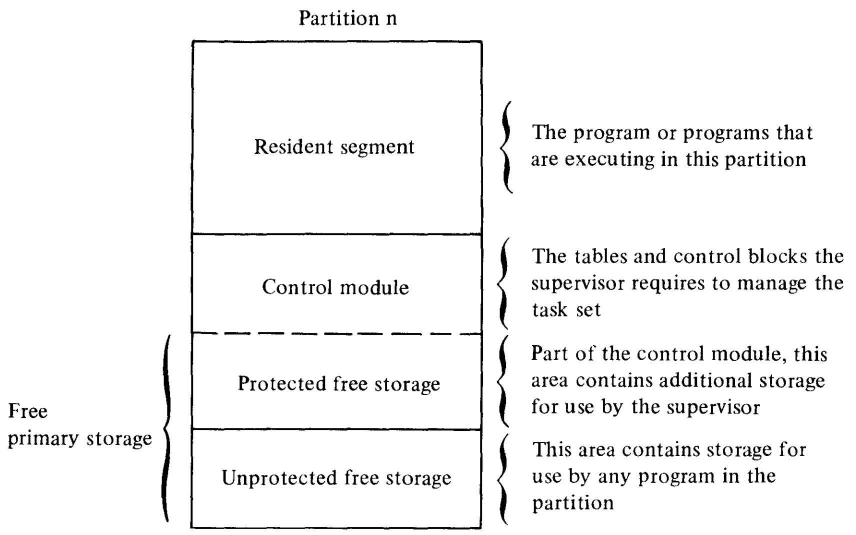
During system generation, you specify the size of the system partition and the size, number, and optionally, the location of each user partition.

Note. If you do not specify a location, the next available location (on a 2KB boundary) after the previously defined partition is assigned.

Your specifications are stored in the IPL options data set, and they can be modified anytime before IPL. To change this information, either repeat the partition-definition step of system generation, or enter the change into the IPL options data set through the text editor (see Chapter 6 for a description of the text editor).

Free Primary Storage

Within a partition, there can be some *free primary storage* — free storage that is available for use by the task set (the program or programs) executing in that partition. Free primary storage is divided into *protected free storage* and *unprotected free storage*. The supervisor allocates free primary storage in response to your requests and system requests.


Partition n
Resident segment
The program or programs that are executing in this partition
Control module
The tables and control blocks the supervisor requires to manage the task set
Protected free storage
Part of the control module, this area contains additional storage for use by the supervisor
Free primary storage
Unprotected free storage
This area contains storage for use by any program in the partition

When two or more blocks of free primary storage are contiguous and of the same type (protected or unprotected), they are combined to make the largest possible blocks of free storage available at all times. To keep free storage from becoming too fragmented, the supervisor satisfies storage requests by allocating the block of storage that best fits the size requirements, not necessarily the first block that fits.

Storage Protection (4955 Processor *Only*)

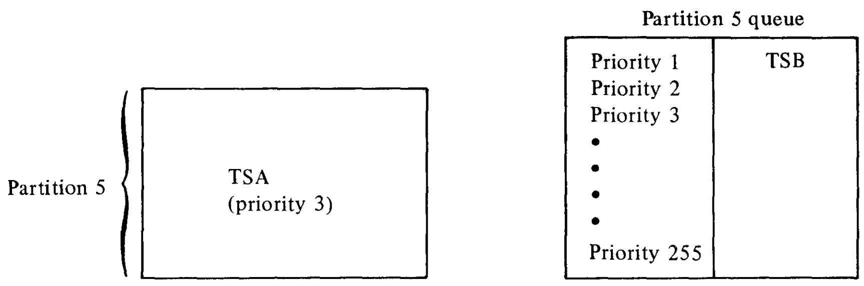
The following can be write-protected by the hardware:

- The entire system task set
- The control module of each task set, which can include a portion of free storage in each partition

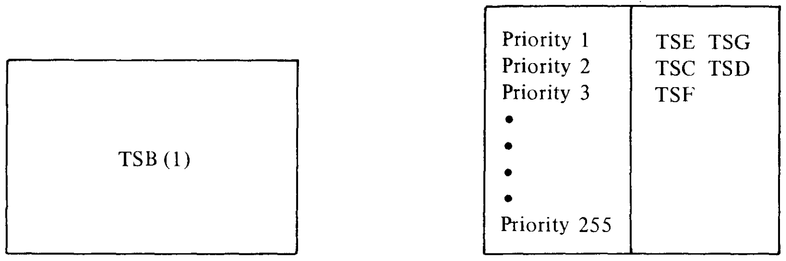
Task Sets

A *task set* is a named collection of programs, data, control blocks, and work areas, forming a related set of work.

Only one task set can execute in a partition at a time. Each partition has a queue of task sets waiting to execute in that partition in first-in-first-out (FIFO) order within priority. Queue priorities range from 1 to 255; 1 is the highest priority. When you place a task set in the queue, you specify the queue priority and the partition in which the task set is to execute. Here is an example:
A task set (lets's call it TSA) is executing in partition 5. TSA has a queue priority of 3. Suppose that TSB is queued for partition 5 with a queue priority of 1.



When TSA has completed, TSB begins executing in partition 5, because it is the only task set queued for that partition. Now, suppose that TSC and TSD are queued with priority 2, TSE and TSG with priority 1, and TSF with priority 3.



Assuming that no more task sets are queued for partition 5, the queued task sets are executed in the order TSE, TSG, TSC, TSD, then TSF.

Creating a Task Set

You create a task set by performing these steps:

1. Code your application programs in assembler language, FORTRAN, or PL/I.
2. Enter your programs through the text editor, then assemble or compile them.
3. Through the application builder, identify the programs that belong in a specified task set, and specify control information for the task set, such as:
   - The entry point at which the task set is to gain control (the entry point of the primary program and the primary task)
   - Control block stack sizes
   - Sensor I/O information
   - Binding information

   The primary output of the application builder is a task set load module, which is placed in a disk data set in a volume called a *task set library.*

In your specifications to the application builder, you can optionally specify *binding* to be performed before the task set executes. *Binding* is the association of a task set with the system resources it requires, such as data sets and I/O devices. The actual binding occurs when the task set is *installed* — when you explicitly request that the task set be loaded and started. Binding a task set lessens the overhead needed to prepare a task set for execution. Binding:

- Predefines tasks by allocating control blocks and task work stacks
- Predefines queues by allocating control blocks and preopening disk data sets for disk queues
- Establishes the conditions under which scheduled tasks are to execute, by updating the system scheduler table
- Preopens data sets

Only those tasks, queues, and data sets that belong to the task set can be bound; any that belong to the shared task set must be bound to the shared task set.

A task set must be prepared for a particular partition, because it is not relocated when it is loaded. A request to execute a task set can be from an operator request or from another program, and the request must specify the partition in which the task set is to execute. A task set can be requested to execute at a specific time or as the result of a process interrupt. At that time, the requested task set is initiated if the partition is available; otherwise, the request is queued, by priority, for the partition. When the partition becomes free, the highest-priority request is initiated and owns the partition until it is done. There is no priority of task sets based on partitions.

There are three types of task sets: *system, user,* and *shared.*

System Task Set

The system task set contains the programs, data, control blocks, and work areas that make up the supervisor.

The system task set executes in partition 0 and resides in protected storage if your machine is equipped with storage protection. Programs in the system task set execute in supervisor state.

User Task Set

A user task set contains the programs, data, control blocks, and work areas that perform a related set of operations for an application.

A user task set executes in a user partition (partition 1—15), and its control module resides in protected storage unless you specify otherwise during task set preparation. The remainder of the task set resides in unprotected storage.

Shared Task Set

A shared task set is a special type of user task set that can contain all the entities of a user task set (such as tasks and programs). It can also contain events, queues, serially reusable resources, and a global area to be shared across user task sets. Task sets using the resources of a shared task set must be resolved against the shared task set by the application builder. You can generate more than one shared task set, but only one can execute at a time.

Tasks. If any tasks are to be started in the shared task set, a primary task must have been started at task set initiation. The primary task of the shared task set terminates only as the result of a request to stop the task set.

Programs. A shared task set need not contain any programs. Programs included in the shared task set can be called by other programs within the shared task set. They can also be called by other task sets that have been resolved against the shared task set by the application builder. A program in the shared task set cannot call a program outside the shared task set.

Events. You can define events in the shared task set to synchronize tasks in different task sets. When an event in the shared task set is used by another task set, the event control block is allocated either in the partition of the task set that defines it as a permanent event, or in the partition of the task set that issues the wait request or is being posted when the event is temporary.

For more information about events see "Events," later in this chapter.

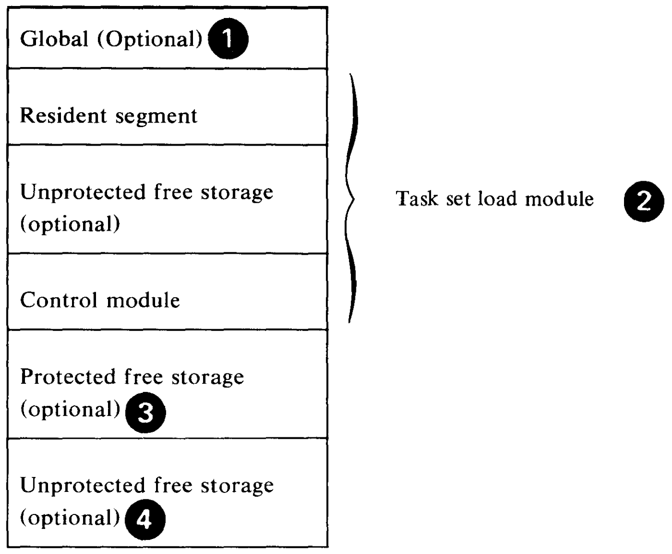
Queues. Queues in the shared task set can be defined either by tasks in the shared task set or by a using task set. The queue control blocks are allocated from the partition of the shared task set.

Global Area. The task set global area is an optional, variable-length, uninitialized data area addressable by all programs executing in the shared task set or by all programs in any task set that is using the shared task set.

Resources. Serially reusable resources can be defined in the shared task set and can be requested by tasks in a using task set. For more information on this topic, see "Serially Reusable Resources," later in this chapter.

Note. Timers and data sets cannot be shared, but they can be included in the shared task set for use by programs executing under tasks in the shared task set *only*.

Format of a Task Set (in processor storage)



Global (Optional) 1
Resident segment
Unprotected free storage (optional)
Control module
Protected free storage (optional) 3
Unprotected free storage (optional) 4
Task set load module 2

1 **Global.** The task set global area is an optional, variable-length data area addressable by all programs executing in the task set. If a task set issues a request to link to another task set—to terminate itself and execute another task set in the same partition—the common global portion is maintained for the linked-to task set.

2 **Task set load module.** The task set load module is in absolute format and contains:

- The *resident segment*, which contains the programs and data that remain in storage for the duration of the task set, plus any overlay areas or subpools required for the task set.
- *Unprotected free storage*, if needed to align the control module on a 2KB boundary. This storage is generated when, in your instructions to the application builder, you specify the PROTECT option for the control module. This storage is available as free storage in the partition.
- The *control module*, which is a set of tables and control blocks required for managing the task set, including:
  - The task set control block
  - Data-set-definition list of data sets that you specify are to be prebound
  - Preallocated control block stacks
  - The variable-control-block area, which is an area of storage available for variable-length control blocks (for the supervisor and data management)

You specify whether the control module resides in protected or unprotected storage. If the control module is in unprotected storage, it is not aligned on a 2KB boundary and is not a multiple of 2KB.

3 **Protected free storage.** This storage follows the control module and is the storage residue, if any, required to protect the control module. If the control module is not protected, or if it is a multiple of 2KB, this residue is not needed. If this storage is needed, it is combined with the variable-control-block area to be used for additional protected storage.

4 **Unprotected free storage.** This is the storage remaining in the partition after any protected free storage or after an unprotected control module.

Reusable Task Sets

In your instructions to the application builder, you can specify that any task set is reusable. A task set should be identified as reusable only if all programs in the task set are either reenterable or reusable.

Note. *Reenterable* is the attribute of a program that allows the same copy of the program to be used concurrently by two or more tasks. *Reusable* is the attribute of a program that allows the same copy of the routine to be used by another task after the current use is concluded.

If a reusable task set is encountered at task set termination, and the next task set ready for execution in that partition is the same task set, the task set being terminated is restored to its original status (its status when it was loaded into storage). As a result, the I/O operations and storage initialization required to prepare the task set for reexecution are bypassed.

Note. A task set that abnormally terminates is not reusable.

Task Set Execution

A task set is first queued to a partition, then it is loaded into storage and executed. This process can be started as a result of:

- A direct request from the system operator or another program in the system
- Scheduling based on the time of day, one or more time intervals, or a process interrupt

The system scheduler table contains entries that indicate task sets to be queued for execution based on these types of conditions:

- Time of day—the task set is queued at the specified time of day
- Regular interval—the task set is queued at the specified interval for the specified number of times
- Process interrupt—the task set is queued at every process interrupt for the device and bit(s) specified

Each entry in the system scheduler table corresponds to a condition number. When a condition is satisfied, the task set associated with that condition is queued for execution. The same task set can be associated with more than one condition number.

Rollout/Rollin

A task set executes until termination unless (1) the partition is identified as a rollout/rollin partition, (2) the executing task set can be rolled out, and (3) another task set is queued to the partition with a queue priority higher than that of the executing task set.

Rollout discontinues execution of a task set and copies its executing environment to a disk data set. *Rollin* copies the disk data set back into storage and resumes executing the task set.

At system generation, you can define one user partition as the rollout/rollin partition. This information is stored in the IPL options data set. You can change it either by repeating the partition-definition step of system generation or by entering the change into the IPL options data set through the text editor. The rollout/rollin partition is the only partition in which rollout/rollin can occur.

If the task set currently running in a rollout/rollin partition can be rolled out, and a higher-priority task set is queued to the partition, the current task set is rolled out. The higher-priority task set executes to completion. The original task set is rolled back in and resumes executing as soon as there are no higher-priority task sets queued to the partition.

Note. Once a task set is rolled out, no subsequent rollout can occur until the rolled-out task set has been rolled back in.

During system generation, you identify one partition as the rollout/rollin partition. Task sets that execute in the rollout/rollin partition must be identified as rollout/rollin if they are to be rolled out; any user task set except the shared task set can be designated rollout/rollin. Program preparation facilities and system utilities are designated rollout/rollin.

Even if a task set is designated rollout/rollin, you can temporarily prohibit rollout/rollin. While rollout/rollin is allowed the rollout/rollin task set *cannot*:

- Use timer services—if timer services are in use when the task set is rolled out, the task set is abnormally terminated.
- Connect to interrupts—if the task set is connected to interrupts when the task set is rolled out, the task set is abnormally terminated.
- Pass addresses or names to other task sets that might attempt to gain access to or modify the contents of the partition while the task set is rolled out; violations could produce unpredictable results in the preempting task set before the preempted task set is rolled back in.

If a task set that uses the shared task set can be rolled out, you must prohibit rollout:

- When the rollout/rollin task set is waiting on an event in the shared task set that is to be posted by another task set
- When the rollout/rollin task set is either defining a queue that resides in the shared task set or processing such a queue by:
  - Waiting for addition of a queue element
  - Locating a queue element
  - Removing a queue element

  Note. It is not necessary to prohibit rollout to add an element to a queue in the shared task set as long as the COPY option is specified to copy the element to the shared task set partition.
- When requesting a serially reusable resource that resides in the shared task set—rollout must be inhibited until the resource is released

A task set is not rolled out until all outstanding I/O requests have completed, even if you have not prohibited rollout.

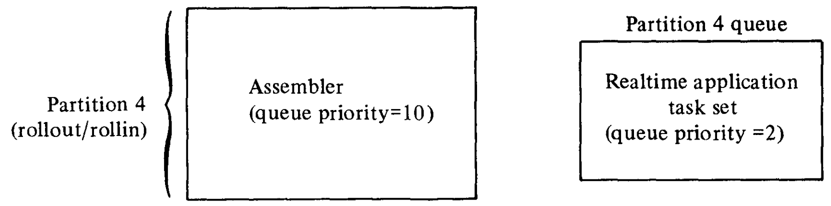
Foreground/Background

A possible use for the rollout/rollin partition is to create a foreground/background situation—run both background (or batch) programs and foreground (or realtime) programs in the rollout/rollin partition.

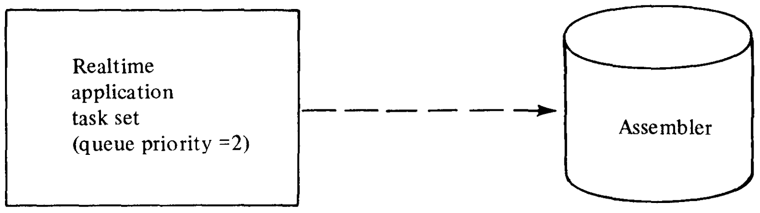
Possible background programs are the program preparation facilities:

- Text editor
- Macro assembler
- Application builder
- FORTRAN compiler
- PL/I compiler

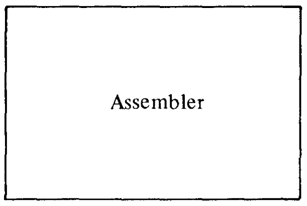
Program preparation can run concurrently with realtime applications. If the background partition is needed for a realtime task set, the background task set is rolled out, the realtime task set executes to completion, then the background task set is rolled back in and continues execution. Here is an example: Partition 4, in this example, is the rollout/rollin partition, and the assembler is running as a rollout/rollin task set (with queue priority 10) in partition 4. A realtime application task set (with queue priority 2) is queued for execution in partition 4.


Partition 4 (rollout/rollin)
Assembler (queue priority=10)
Partition 4 queue
Realtime application task set (queue priority =2)

Because the realtime application task set has a higher priority, the assembler task set is discontinued and is rolled out to a disk data set. The realtime task set is loaded into partition 4 and executes to completion.



When the application task set has completed, the assembler task set is rolled back into partition 4 and continues executing, provided there are no higher-priority task sets queued for execution in partition 4.



Task Set Termination

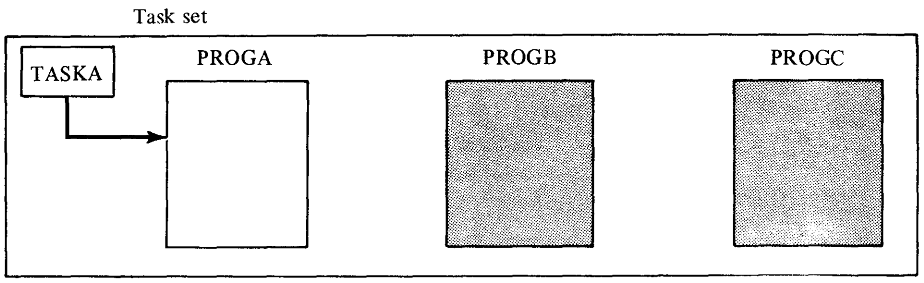
A task set terminates in one of two ways. It can terminate normally, when the primary task terminates normally, or it can terminate abnormally, as the result of either:

- An explicit request to terminate the task set abnormally. In this case, a storage dump is taken if specified on the task set termination request.
- Abnormal termination of the primary task. In this case, a storage dump is taken only if requested as a result of primary task termination.

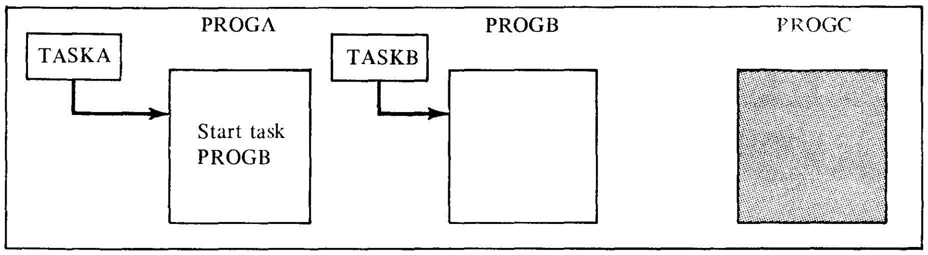
Tasks

A *task* is a single thread of execution through one or more programs. It is defined at execution time, when either the supervisor or a program in the task set issues a request to start another program (or programs) executing concurrently as a task. For example:

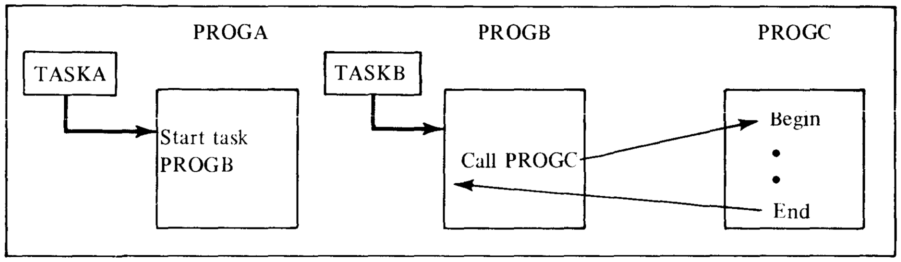
Partition n contains a task set. The task set contains three programs—PROGA, PROGB, and PROGC. TASKA has been previously defined as the *primary task* (with PROGA as the primary program), so it receives control when the task set is loaded (the *primary task* is the one that is started automatically when the task set is started).



In this example, suppose that PROGA issues a request to start TASKB (with PROGB as the primary program) executing as a *secondary task* (a secondary task is one that is started by a request from another task).



PROGB begins executing concurrently with PROGA. Now suppose that PROGB issues a CALL to PROGC.



PROGC immediately gains control, executes to completion, then returns control to PROGB.

Task Priorities

In your request to start a task, you also specify its software priority within the interrupt level. In that way, you can control the order in which tasks are *dispatched*, or prepared for execution. Within each hardware interrupt level, there are 255 priority levels (numbered 1 through 255; 1 is the highest). This means that if, on a certain interrupt level, two or more tasks are requested simultaneously, the one with the highest priority is dispatched and executed first.

Terminating Tasks

A task can terminate in one of three ways:

- Normally, when the initial program in the task returns control to the supervisor
- Abnormally, as the result of either an explicit request from the same task or another task, or termination of the task set
- Abnormally, as the result of a program check or soft exception

Note. When the primary task terminates, the task set automatically terminates.

The termination process occurs under the terminating task, and therefore has the same priority as the terminating task.

Normal Termination

During normal task termination, the system automatically:

- Closes data sets not closed by the task
- Deletes timers, events, and queues not deleted by the task
- Relinquishes control of resources owned by and requested by the task
- Frees storage allocated for the work stack and floating-point-register save area
- Deallocates the task control block and execution request blocks
- Disconnects interrupts

If the terminating task is prebound, the task control block, the floating-point-register save area, and the work stack are not deleted in case a request is issued to start the task again.

Abnormal Termination

During abnormal task termination, the system executes the error-exit subprogram, if one is active for the task. If the error-exit subprogram returns with the option to resume execution, the task resumes at the instruction indicated.

If task execution does not resume, and a storage dump is requested, primary storage is copied to the dump data set, and normal termination processing is invoked.

Programs

Program management controls two types of programs—problem programs and supervisor programs.

Problem Programs

Problem programs can be either (1) storage-resident programs, which remain in storage from the time the task set is loaded until task set termination, or (2) overlay programs, which reside on disk or diskette until loaded into an overlay area within the partition. In your instructions to the application builder, you specify whether a program is storage-resident or an overlay.

There are two types of problem programs:

- *Primary program*—the first program of a task
- *Secondary program*—called by another program

At the beginning of a primary program, you must indicate its *attributes*—whether it is reenterable, reusable, or nonreusable. Reuse of these programs is managed by the supervisor, based on the attributes you specify. The primary program is storage-resident, and it can call overlays. When the primary program terminates, the task terminates.

The request to execute a secondary program does not indicate whether the program resides in storage or on disk; if the program is an overlay, the request to execute causes the program to be loaded into a predefined overlay area. Requests to execute secondary programs can pass parameters to the requested program.

A secondary program can be storage-resident, or it can be an overlay that, when called, is loaded into the overlay area of the primary program. A secondary program that is an overlay cannot call another overlay.

The supervisor does not manage the reusability of a secondary program if the program is resident, because resident programs are assumed to be reenterable. However, if the program is an overlay, the overlay manager reuses the overlay program if it receives a request for an overlay program that is already in the overlay area. Overlay programs are assumed to be reusable.

Supervisor Programs

A supervisor program can be either storage-resident or a disk-resident transient. The request to execute a supervisor program does not indicate whether the program resides in storage or on disk.

Each supervisor program has a prolog that identifies the entry point, program work space requirements and number of parameters to be passed to the program. Supervisor program management automatically resolves the addresses of parameters specified when a supervisor program is invoked; a list of these addresses is built in a dynamic area to allow the supervisor program to be reenterable.

When a supervisor program is invoked, the environment (for example, the general-purpose registers) is automatically saved upon entry, then is restored before a return is made to the invoking program. A return code can be returned to the invoking program.

A transient program can be loaded into either a fixed or a dynamic transient area. It can be loaded into a fixed transient area when it is invoked, then deleted upon completion if no other requests for the transient are unsatisfied. A transient program can be loaded into a dynamically allocated transient area before it is invoked, and it can remain in storage until (1) an explicit request is made to delete it, and (2) all requests to execute the transient have been satisfied.

Transient programs must be refreshable. At system generation, within system-established limits, you can specify what is transient or resident; however, you cannot specify whether transient programs are loaded into a fixed or dynamic load area—that is determined by the system. Supervisor programs can be invoked explicitly or as the result of an SVC.

Although IBM provides most supervisor routines, you can, if you wish, write your own supervisor routines. All supervisor programs operate in supervisor state.

Parameter Lists

Parameter lists pass information between programs. There are two types of parameter lists, and the type you choose depends on the types of programs that are to use it:

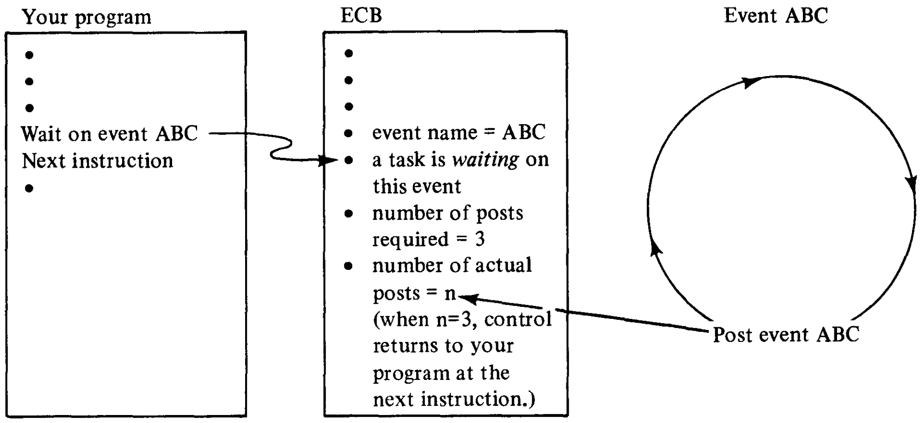
- *Basic* parameter lists are for problem programs, and for supervisor programs that are invoked by CALL instructions.
- *Extended* parameter lists are for supervisor programs and for any routines that are invoked by supervisor call (SVC).

Events

You can synchronize the execution of tasks by (1) identifying events that are of significance to the tasks, (2) specifying the points where execution cannot continue until an event occurs (*waiting* on the event), and (3) signalling the waiting task when the event occurs (when the event has been *posted* the specified number of times).

An event is controlled through an event control block (ECB), which contains such information as the name of the event, whether it is being waited on, how many times it must be posted for the event to complete, and how many times it has actually been posted. Here is an example of an ECB in use:

Your program first issues a WAIT instruction for event ABC, causing the ECB for that event to indicate that a task is waiting for event ABC. Each occurence of event ABC issues a POST to the ECB. When the ECB has been posted the specified number of times, your program resumes at the next instruction after the WAIT.


Your program
•
•
•
Wait on event ABC
Next instruction
•
ECB
•
•
•
• event name = ABC
• a task is *waiting* on this event
• number of posts required = 3
• number of actual posts = n
(when n=3, control returns to your program at the next instruction.)
Event ABC
Post event ABC

An event can consist of from 1 to 255 posts, depending on how you define it. For example, an event can indicate that:

- An I/O operation is complete
- A task has reached a certain point in its processing
- An element has been added to a queue
- An occurrence has happened a specified number of times

Each event is known by its name. The meaning of an event name is determined by the event definition and its use by the communicating tasks. An event must be referred to by the same name by all tasks that either wait on the event or post the event. Events can be defined as:

- Permanent events, which means that the ECB exists until explicitly deleted or until termination of the task
- Temporary events, which means that the ECB is allocated when it is referred to and exists until the event is complete

A task can suspend execution until the completion of either one specific event or any one of a list of events. If a list of events is specified, the task is notified which event in that list completed to cause the task to resume execution.

When an event completes it is automatically reset; that is, the ECB is set to *not waiting*, and the post count is reset.

Queues

You can define two types of data queues:

- Storage queues—they reside completely in main storage
- Disk queues—definitions of these queues reside in main storage; control information and data elements reside either in main storage or in a direct-access data set, depending on the priority of the element.

Storage queues and disk queues can be defined as either public or private. A *private queue* is associated with the task that defined it. Only the task that owns a private queue can locate elements on the queue or remove elements from the queue. A *public queue* can be processed by any task in the task set.

The task that processes a public queue is responsible for serializing access to the queue.

The size of each queue element is specified when the element is added to the queue, and a queue can contain elements of varying sizes. If requested, an event is posted when an element is added to a queue.

Storage Queues

The maximum number of elements for a storage queue can be specified when the queue is defined, or the queue can be defined to use whatever storage is available. When the queue is full, any attempt to add an element is unsuccessful.

Elements in a storage queue are maintained in FIFO sequence within priority. Each element has a priority, which is specified when the element is added to the queue.

Disk Queues

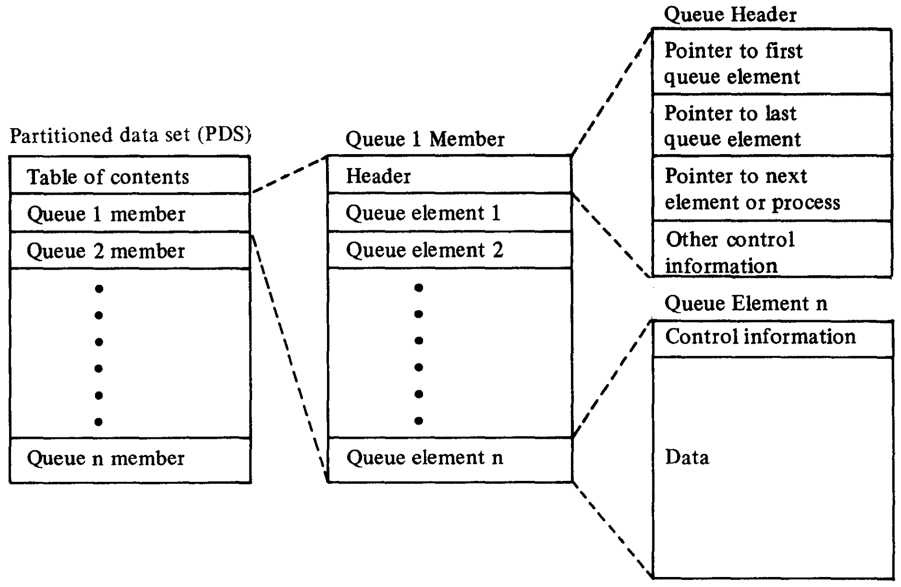
A disk queue is a member of a partitioned data set (PDS); the queue name is the member name. One or more queues can reside in the same PDS, and a system can contain one or more PDSs that contain queues. A header of control information is kept with each queue in its appropriate member. This control information is updated on disk whenever the queue is updated (each time an element is added or deleted) and whenever an element is located.

The disk queue header contains sufficient information to restart the disk queue at the latest point of processing. When defining the queue, you also define the type of restart capability you want:

- *Warm restart*—the queue is restarted containing any elements it contained when it was last used
- *Cold restart*—the queue is restarted containing no elements

There are two types of disk queue elements—normal and priority. *Normal* queue elements are maintained on disk in a FIFO sequence. When the queue becomes full, there is a wraparound option that allows a new element to be written over the oldest element on the queue. *Priority* queue elements are maintained in main storage in FIFO sequence within priority.

In your definition of a disk queue, you include the maximum size of an element to be added to the queue and the name of the PDS member to contain the queue. Elements processed into and out of the queue can be of variable length, but records written to the queue data set are all the size of the maximum element. The queue element and its related control information are written to the data set.



You must create the PDS for a disk queue before you define the queue. The attributes of a data set member are:

- The member name is the queue name.
- The record format is either fixed or fixed blocked.
- The record length is the same as the maximum element size that you specified when you defined the queue, *plus* the length of the control information.
- The number of records in the data set must be greater than or equal to the maximum number of elements defined for the queue, plus one.

All queues in a single PDS must have the same maximum element size.

When you process a disk queue, you must first locate an element, then gain access to it. Only the last element located can be processed.

Timers

The services of timer management control logical timers, the time of day (maintained in the system), and the date (also maintained in the system). These services are needed for time- or date-dependent operations.

There are two physical timers in a hardware timer attachment. All logical timers use one physical timer—the other physical timer in the hardware attachment is used for time-of-day services.

The supervisor timer services (optional) require a dedicated timer attachment. Your applications can use other timers by means of data management functions.

Logical Timers

A logical timer is the software interface to a physical timer. There are two categories of logical timers:

- Asynchronous timers—normally used for timeouts, these timers run concurrently with the tasks that use them.
- Synchronous timers—normally used for delay operations, these timers cause the tasks that use them to wait for a specified time interval.

Time-of-Day and Date Services

The time of day and the date are maintained in the system; you can gain access to them through your program or operator commands.

The system maintains the time of day in a physical timer unless, during system generation, you specify that time-of-day services are to be omitted. Time of day is available in hours, minutes, seconds, tenths of seconds, and milliseconds. Time references are the time since midnight. The supervisor compensates for drift in the physical timer by adding a compensation factor while maintaining the time of day.

If you do not use the physical timer, you can supply a program to maintain the time of day by means of any periodically interrupting source. The resolution of time of day then depends on the accuracy of the interrupting source.

The date is available in either Julian or Gregorian format. If you choose Gregorian, you must specify which format you prefer—MMDDYY or YYMMDD.

Serially Reusable Resources

A *serially reusable resource* is that which can be serially reused by numerous requesters; for example, programs, events, queues, and data sets can be considered serially reusable resources.

You request that the supervisor serialize access to logical resources that can be requested asynchronously from different tasks. To do this, you designate each resource as a serially reusable resource and assign it a symbolic name. Each access to the resource is preceded by a request for that resource name, then followed by a release of that resource name.

When a task requests access to a serially reusable resource, the resource is marked *busy* and subsequent requests from other tasks are not granted until the resource is released from the task that first requested it. Requesting tasks wait until the resource is no longer busy.

Interrupts

Interrupt management processes I/O interrupts (data-processing, timer, and sensor-based process interrupts) and class interrupts.

I/O Interrupts

The supervisor processes priority interrupts from data-processing I/O devices, timer I/O devices, and sensor I/O devices. When an I/O interrupt occurs, it activates the appropriate device interrupt service task. An interrupt service task can be associated with more than one device.

Interrupt service tasks are dispatched in supervisor state and are part of the system task set. All control blocks generated and used by I/O interrupt management are within the system task set.

Class Interrupts

Class interrupt handlers are storage-resident routines that process the interrupts that result from error or exception conditions detected by the hardware. They handle these types of interrupts:

- Machine check
- Program check
- Power/thermal warning
- Programmer console
- Trace
- Soft exception

- SVC

A class interrupt automatically disables all priority interrupts. Also, programmer-console and power/thermal interrupts are disabled after a class interrupt. The class interrupt handlers enable interrupts as quickly as possible.

Class interrupt handlers always run under a task. If no task is active on the level when a machine check, power/thermal warning, or programmer-console interrupt occurs, the supervisor uses a dummy system task. If another class interrupt occurs, with no active task, while the dummy task is in use, the second interrupt is ignored. The exception to this procedure is the SVC interrupt, which is valid only when a task is active.

The supervisor intercepts all class interrupts. When you generate your system, you might want to include your own programs, which are invoked after the system has handled these interrupts:

- Machine check
- Power/thermal warning
- Programmer console
- Trace

Your program resides in the system partition and gains control in supervisor state. The supervisor passes some status information to your program, and enables all interrupts. Your program has no restrictions as to system functions available to it. It might prepare for system shutdown, or it might attempt to reload and restart.

Errors

Error management services assist in the determination and correction of errors detected by hardware and software. Error management comprises:

- Error logging and reporting
- Abnormal termination of a task
- Copying primary storage to a data set for later retrieval
- Task error exit
- Displaying and patching storage
- Attended/unattended status indicator
- System reload and restart
- System termination

Error Logging and Reporting

There can be up to 16 error logs defined during system generation or execution. Each log has an ID of 0—15, with 0 reserved for the system error log. During system generation, you specify the maximum number of error logs.

The purpose of the system error log is to record processor and I/O errors for the devices supported by the supervisor. Software errors or other information can also be recorded in the system log or in another log in the system. Error logging and reporting:

- Records certain hardware status information
- Builds a printable error record
- Prints the error record at the operator station
- Writes the error record to the error log

The system error log is optional. If it is not defined and activated, requests to write to the system log are ignored; however, error log messages directed to the operator station are always printed.

An error log resides either on disk or in storage. You can use a utility to get a hexadecimal dump of an error log. The individual records in a log can be retrieved through your program.

Abnormal Termination of a Task

A task can terminate abnormally as the result of either supervisor action or your request. A request to terminate a task is always considered an abnormal termination unless otherwise specified. The termination request can optionally request that a storage dump be taken. If the termination is the result of a supervisor action, a dump is always taken.

If a task is using the shared task set, it can terminate any task in the shared task set. If it is not using the shared task set, a request to terminate another task can be made only within the same task set. If the terminating task is the primary task within the task set, the task set is terminated.

Abnormal termination can pass control to a task-error-exit routine, if you have specified one. This routine can optionally indicate that the task resume execution unless (1) the task is terminating as a result of a machine check or program check, or (2) the task set in which the task is active is abnormally terminating.

Copying Primary Storage (Storage Dump)

When a task or task set terminates abnormally, all of primary storage can be dumped to a data set. The dump data set can later be retrieved, formatted, and printed with a utility program.

Task Error Exits

You can specify a task error exit at any point during task execution, indicating a subprogram to gain control if the task terminates abnormally. The subprogram has the option of resuming the task or continuing the termination process.

You supply the task-error-exit subprograms. In your instructions to the application builder you must include these subprograms with the other programs in the task set when the task set is created.

Displaying and Patching Storage

The operator can display up to 56 bytes of contiguous storage and patch up to 20 bytes of contiguous storage.

Attended/Unattended Status Indicator

This indicator reflects whether an operator and a system console are present. The error processing routines check this indicator to determine whether operator intervention can be expected.

System Reload and Restart

The system-restart process terminates the current supervisor and loads the restart supervisor, then starts the system initialization process. The system-restart process is primarily used to restore the operating system after a serious error has made the current system unable to continue. Restart can, however, be issued at any time to load the restart supervisor.

System Termination

System termination handles system error conditions (such as supervisor or hardware failures) that make continued operation of the system impossible.

When the system termination routine gains control, it attempts to write to the system error log. Then, a termination exit program is invoked, if one is available, and it is passed the completion code indicating the cause of failure. The exit routine can then either return control to the routine in error, restart the system, or terminate the system.

You supply the termination exit program, and it must reside in the system task set. It must be included during system generation.

Related Publication

When it is available, the *IBM Series/1 Realtime Programming System: Macro User's Guide—Supervisor* will deal with the topics in this chapter at the assembler-language level.

Chapter 3. The Operating System: Data Management

Data management moves information between processor storage and external devices, maintains the data on those devices, and controls those devices. Data management is divided into two categories—*data set management* and *device management*.

Data set management handles data sets, which are named collections of data residing on a device. It has:

- Three levels of access—*basic, physical,* and *logical*
- Two access methods—*sequential* and *direct*
- Three data set organizations—*consecutive, random,* and *partitioned*

Access levels, access methods, and data set organizations are discussed later in this chapter.

Device management handles physical devices, and handles errors and interrupts from them. Device management is described later in this chapter.

Data Set Management

Data set management maintains permanent data sets and gains access to all data sets on all devices. All access to data sets or devices (except for basic access) is through the data set definition (DSD), which describes the data set or device and how a program can gain access to it. A DSD might be in the using program or in a DSD table data set in the task set library. You can change a data set's attributes in the DSD through a utility or a job stream processor statement, thereby avoiding the need to change any code in your program.

Direct-access sets can be of two formats—*basic exchange format* and *extended format*. A basic exchange data set is one that is formatted to certain standardized specifications so that the same data set can be used with other computer systems. (For the specifications of a basic exchange data set, refer to the *IBM Series/1 Realtime Programming System: Macro User's Guide—Data Management.*) Basic exchange data sets can be on diskette only, while extended format data sets can be on disk or diskette. In defining an extended-format data set, you can specify up to four levels of qualification—*device, volume, data set,* and *member.* A disk or diskette using the extended format can contain one or more logical volumes, each of which can contain several data sets, which might contain members.

When you want to use a data set or device, you must first open it. Opening a data set or device supplies data set management and device management with the information they need for access to that data set or device. When access to a data set is complete, a close request breaks the connection between the data set and the requesting task.

Data Set Organizations

There are three types of data set organization—consecutive, random, and partitioned. These organizations are available at the physical and the logical access levels. Random and partitioned organizations are available on direct-access devices only. Consecutive organization is available for any device. The data set organization for a data set is determined when the data set is created or opened. You choose the type of organization, depending on the application of the data set.

Consecutive Organization. In a data set that has consecutive organization, a program has serial access to blocks and logical records. Records in these data sets can be either fixed- or variable-length. Individual records can be grouped to form blocks, and records can span blocks. A program can use the sequential access method to retrieve records in a consecutive data set, or it can use the direct access method if the data set resides on a direct-access device or display station and the records are fixed-length.

Random Organization. In a data set that has random organization, a program can retrieve blocks and logical records in any order. Only direct-access devices (such as disks and diskettes) can use random organization. Random data sets have fixed-length records. Individual records can be grouped to form blocks, and records can span blocks. A program can use either the sequential or the direct access method to retrieve records.

Partitioned Organization. A partitioned data set *contains* data sets and has a table of contents that describes and locates the data sets within it. A subsidiary data set is called a *member*. A member need not have the same attributes as the other members. A partitioned data set can be on a direct-access device *only*.

The table of contents of a partitioned data set describes the members of the data set. It contains indexes, which contain the names of the members and the attributes of the members.

Levels of Access

There are three levels of access to a data set—basic, physical, and logical The level chosen is determined by the instruction used to gain access to the data set.

Basic (EXIO). Through the basic level, you have access to data on a device by physical record. This type of access is device-dependent and is done at the hardware instruction level. With the basic access level, you have access to any operation the device can perform.

Physical (READ/WRITE). Through the physical level, you have access to data on a device by block, which can consist of one or more physical records. This type of access is valid for any of the three data set organizations.

Logical (GET/PUT). Through the logical level, you have access to data by logical record, with automatic blocking and buffer management. This type of access is device-independent, unless you specify certain device-dependent controls. Logical access is valid for any of the three data set organizations.

Access Methods

Data set management has two access methods—sequential and direct. The access method is determined when the data set is opened. Both access methods can be used at the physical or logical access level. Any I/O device can use sequential processing, but only a direct-access device, such as a disk, diskette, or display station, can use direct processing. You choose the access method to be used depending on the application of the data set.

Sequential Access Method. With the sequential access method, access to data is from the beginning of the data set to the end-of-data indicator. Sequential access is valid for all three data set organizations. Access can be through either the physical or the logical level. Sequential records can be either fixed- or variable-length, and can be blocked and spanned; however, fixed-length and variable-length records cannot be mixed within a data set. For unit-record devices, sequential records can be fixed-length only.

Direct Access Method. With the direct access method, access to data can be anywhere in the data set. The records must be consecutive and fixed-length, and they can be blocked and spanned. Direct access is valid for all three data set organizations, and can be at the physical or logical level. The data set must be on a direct-access device or a display station.

Device Management

Device management handles (1) requests to transfer data to or from an I/O device, and (2) interrupts from I/O devices. To do these things, device management uses a set of device handlers, which are part of the system task set. Device handlers can be started when the system is loaded, or you can start them when you issue the request to start a device.

The following sections briefly discuss the devices handled by device management. You can write your own device handlers and add them to the system task set. For information on doing this, and for information on device handlers for devices other than those supported by device management, see the *IBM Series/1 Realtime Programming System: Macro User's Guide—Data Management*.

Data-Processing I/O

The data-processing I/O portion of device management controls these devices through either a READ/WRITE, GET/PUT, or EXIO interface:

- Operator station (teletypewriter)
- Display station
- Matrix printer
- Line printer
- Diskette unit
- Fixed-disk storage unit
- Timers (other than those used by the supervisor for time-of-day and logical timer services)

Sensor-Based I/O

The sensor-based I/O portion of device management controls these devices through either a READ/WRITE or an EXIO interface:

- Digital input
- Digital output
- Analog input
- Analog output

Timer I/O

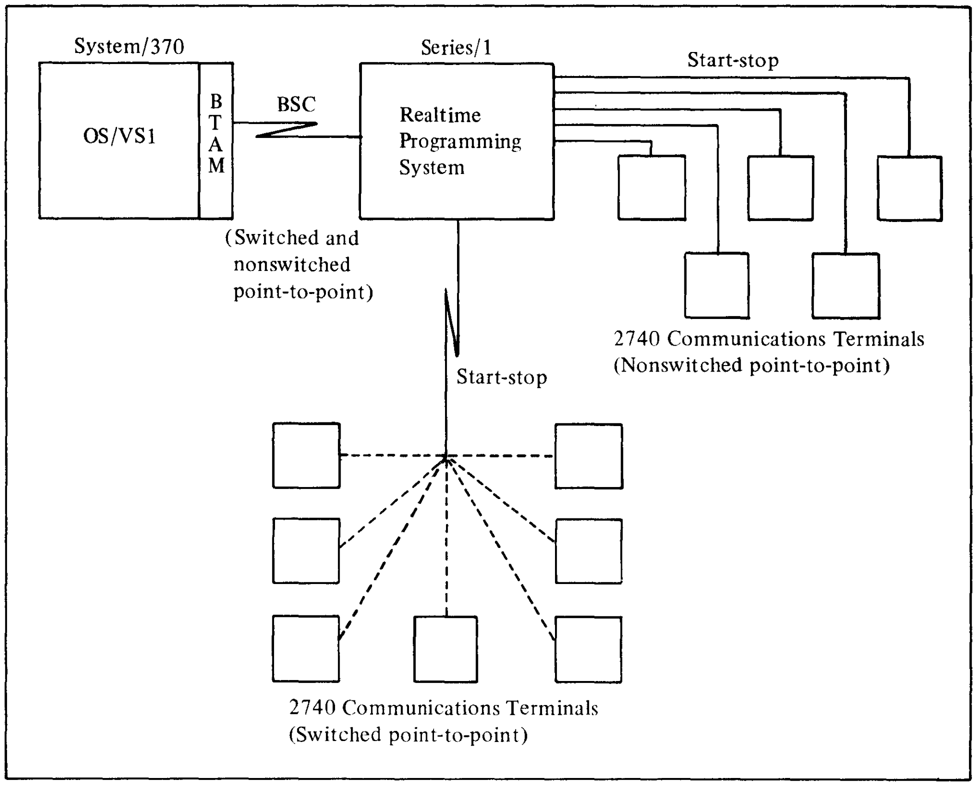
Access to timers is through either a READ/WRITE or an EXIO interface.

Related Publication

The *IBM Series/1 Realtime Programming System: Macro User's Guide—Data Management* will, when available, give detailed information about the topics covered in this chapter.

Chapter 4. The Operating System: Communications

Communications is the part of the operating system that directs the transfer of data between your programs and remote stations. A *remote station* is either a terminal or another computer. Communications handles point-to-point connections between stations that use start-stop and binary synchronous communications (BSC) line control. The following example illustrates a possible communications system.


System/370
OS/VS1
BTAM
BSC
Series/1
Realtime Programming System
(Switched and nonswitched point-to-point)
Start-stop
Start-stop
2740 Communications Terminals (Nonswitched point-to-point)
2740 Communications Terminals (Switched point-to-point)

The communications portion of the operating system:

- Establishes, terminates, and controls access between your programs and remote stations
- Transfers data between your programs and remote stations on point-to-point lines

You can also, through services of the operating system, translate from one transmission code to another, such as from EBCDIC to ASCII.

The stations used by start/stop communications are (1) the 2740 Communications Terminal Model 1 (switched and nonswitched), and (2) the Teletype Model 33/35 (trademark of the Teletype Corporation) or equivalent, nonswitched.

Communications between a Series/1 and a System/370 using OS/VS1 BTAM is supported through BSC line control in point-to-point connections.

During system generation, you supply the information needed to tailor the BSC and start/stop communications system to your requirements.

Configuration of a Communications System

You must describe the configuration of your communications system to the operating system. This includes defining the remote stations with which you intend to communicate and the characteristics of those stations. The operating system treats a remote station as a temporary data set—a data set in which the contents are not saved after transmission.

Opening and Closing Communications Data Sets

Opening a communications data set establishes a connection between your program and a remote station. This connection must be established before you can communicate with that station. Opening a data set:

- Associates the remote station (specified by you) with the line
- Verifies that the current line definition is compatible with the remote station attributes given in the data-set-descriptor control block
- Connects the issuing task to the remote station

A communications data set remains open until you issue the instruction to close it. Closing a data set breaks the connection that was established when the data set was opened. Closing a data set:

- Terminates the connection between the task and the remote station
- Deallocates or disconnects the line from your data set

Processing Communications I/O Requests

Reading from and writing to remote stations (through the READ/WRITE interface) is handled through events. When a read or write operation is issued, the task waits for its completion. When the operation completes, the event is posted.

Before issuing an I/O request, you must define the physical buffer from which or to which the data is to be transferred. The address of the buffer is made available to the operating system when the I/O request is issued.

All data transfers are made directly from the storage buffer to the remote station and from the remote station to the storage buffer. You are responsible for ensuring that the buffers are properly managed. The output buffer must be filled before a write request can be honored.

Support for the Model 33/35 is limited to making and maintaining half-duplex line connections, transmitting user-furnished buffers on the line, receiving data from the line and filling user-furnished buffers, and recognizing user-supplied change-of-direction characters on receive as end of data.

You are responsible for inserting and deleting terminal control characters (such as, for start/stop, carriage return and line feed). On a read operation, data is placed in the storage buffer exactly as it comes across the line, including all the control characters used by the remote station to transmit the message.

Communications I/O Operations

Using read and write instructions, you can do these communications I/O operations:

OPERATION	*BSC*	*START/STOP*
Transmit	X	X
Receive	X	X
Transmit Interspersed with Receive	X	
Receive Interspersed with Transmit	X	
Transmit with Conversational Reply	X	
Receive with Conversational Reply	X	
Transmit Reply	X	
Transmit EOT	X	
Transmit Acknowledgement		X
Transmit Interrupt		X

Transmit. The transmit operation sends data to a remote station.

Receive. The receive operation requests data from a remote station.

Transmit Interspersed with Receive. This operation is initiated when, while you are transmitting data, a remote processor issues a request to send a high-priority message to you. The remote processor issues a reverse-interrupt request. After the data is received from the remote processor, transmission resumes.

Receive Interspersed with Transmit. This operation is initiated when, while receiving data, you wish to send a high-priority message to the remote processor. You issue the reverse-interrupt request. After the data is transmitted to the remote processor, you can resume receiving.

Transmit with Conversational Reply. With this operation, the remote station responds to your transmissions with either a positive acknowledgement or data. If the remote station sends a positive acknowledgement, the transmit operation is posted with normal completion, and another transmission can be issued. If the remote station responds with data, the transmit operation is posted with a completion code indicating that data was received and must be processed.

Receive with Conversational Reply. With this operation, you respond to transmissions from the remote station by sending data. After sending the data, you either transmit another message or wait for a reply from the remote station.

Transmit Delay. With this operation, you can send a delay sequence instead of the next operation.

Transmit EOT. This operation lets you send an end-of-transmission character.

Transmit Acknowledgement. This operation lets you transmit positive or negative acknowledgements. These acknowledgements interrupt the transmission of multiple blocks of data from a terminal to the processor.

Transmit Interrupt. This operation writes a continuous space signal to a remote station and turns the line around. With this operation, you can initialize a remote station.

Online Terminal Tests

With online terminal tests, an operator can test start/stop terminals concurrently with normal operations. The test does not interfere with normal operations. By entering a request from the terminal, the operator can transmit a record to the Series/1 and have it transmitted back to the terminal a specified number of times to verify that messages are being transmitted and received intact. This type of test, commonly called an *echo* test, is available for the 2740 Model 1 communications terminal.

Online terminal testing is optional — you specify it during system generation.

Related Publication

The *IBM Series/1 Realtime Programming System: Macro User's Guide—Communications* will, when available, give detailed information about communications.

Chapter 5. The Operating System: Utilities

The Series/1 utilities are part of the operating system. They are IBM-supplied application programs with which you can easily and efficiently manage data and maintain your system.

Series/1 utilities can be divided into two categories—*stand-alone* and *system* utilities. The stand-alone utilities are loaded into storage from diskette, whereas the system utilities require a disk as the system-resident volume.

The following sections briefly describe the utilities that are available. For details on what they do, how to use them, and how they work, refer to *IBM Series/1 Realtime Programming System: Operator Commands and Utilities.*

Stand-Alone Utilities

The stand-alone utilities are a processor-storage-to-diskette dump, disk initialization, and system build. You load the stand-alone utilities from diskette, and no other program can execute concurrently with them.

The processor-storage-to-diskette dump utility resides by itself on a diskette, and runs on any Series/1 with a diskette unit and at least 16K bytes of storage.

The disk initialization and system build utilities are supplied on a diskette along with a stand-alone utility monitor. You load this diskette to invoke utilities running under the stand-alone monitor.

The disk initialization utility initializes your disk for use with Series/1 and must be invoked before the system-build utility is executed for a new disk device.

Processor-Storage-to-Diskette Dump

This dump utility takes a stand-alone dump of processor storage and registers onto diskette with minimal loss of storage contents. The dump is taken onto the diskette from which the utility was loaded. Only 256 bytes of processor storage—addresses 0 through 255—are lost when this utility executes.

Disk Initialization

Disk initialization has two initialization types—primary initialization and alternate sector assignment.

Use primary initialization for complete initialization when the disk is installed, or when complete reinitialization is necessary. The utility first verifies and corrects sector IDs, then analyzes the disk surface to find defective sectors. When it finds a defective sector, it assigns an alternate sector on cylinder 1, and prints a message at the operator station.

Alternate sector assignment lets you assign alternate sectors for those found to be defective. It also attempts to recover the data from the defective sector and move it to the alternate.

System Build

This utility is the first program to be executed during installation of the operating system, because it builds your disk in preparation for executing system utilities or for system generation.

This utility copies to disk the starter system diskettes shipped from IBM. These diskettes contain libraries needed to load the starter system from the disk and to either begin the system-generation process or execute system utilities. This utility can also be used to restore the starter system and system utilities, and make the system suitable for loading.

System Utilities

The system utilities run under the operating system, and they require a disk as the system-resident volume. When you invoke the system utilities, they are loaded as a task set into a user partition.

System utility commands can be entered by way of an interactive device (such as an operator station) or a noninteractive device (such as a disk). You can enter system utility programs in one of two modes—*single-line* and *prompt/reply*. If you are familiar with the format of a particular command, enter it in single-line mode—enter all the required information in a single statement. However, if you are not familiar with a certain command's format, use the prompt/reply mode. That mode leads you step-by-step through the procedure. If necessary, you can switch from one mode to another between commands.

The system utilities have some special features that make them easier to use:

- *Hexadecimal or decimal numerics*—You can enter a value in any numeric field using either decimal or hexadecimal notation.
- *Cancel*—You can cancel the currently requested utility any time that you are allowed to enter a reply. You can then restart the cancelled utility, start a new one, or end the utility session.
- *Comments*—You can enter comment records, which are not processed as commands or data. This feature is used when commands are being entered from a disk data set.
- *Line continuation*—In single-line format you can, if necessary, continue a command on the next line.
- *Error recovery*—This depends on the mode you are using:
  - In prompt/reply mode, a syntax error is reported immediately—before the next prompt. The utility tells you the reason for the error, then repeats the prompt.
  - In single-line mode on an interactive device, your error is reported, and the utility requests that you correct and reenter the entire command. On a noninteractive device, the error is reported and the utility session terminates.

Compress

This utility consolidates all available free space into one contiguous area within the specified partitioned data set or volume. It lets you compress in-place any partitioned data set or volume that contains 500 or fewer table of contents entries. The specified data set is opened by the utility for exclusive use only.

Note. If you do not wish to compress in-place, or if you have more than 500 members in the data set to be compressed, use the COPY UNIT utility.

Copy

This utility transfers an image from one location to another. The locations can be on the same or a different device or device type. For example, you can copy a data set from a disk to a diskette. If the member you are copying to does not currently exist and space is available, the member is created for you, using the attributes of the member that you are copying from. This utility has four subfunctions:

- UNIT—You can copy an entire volume, data set or member, or device (disk or diskette) by specifying the unit to be copied. You have the option of compressing the target data set or volume.

- ADD—You can add records to the end of an existing consecutive data set.
- MULT—By specifying a volume, data set, or member prefix, you can copy all elements that begin *with* the prefix to a target data set; or, you can copy all elements *without* this prefix to the target data set
- DISKETTE—You can copy the entire physical contents of one diskette to another diskette.

Notes.

1. Unique diskette information is not copied.
2. This copy feature uses either:
   - Two or more diskette units (if your hardware configuration has them) or
   - A diskette and a disk (if you have only one diskette unit).

Define

This is a multifunction utility—the following list briefly discusses what it can do:

Define an Extended-Format Disk or Diskette. With this option you can define an extended-format disk or diskette containing logical volumes. This utility must precede your actually creating the logical volume on the device.

Create a Consecutive or Random Data Set or Member. This option creates a consecutive or direct data set or member on an extended-format disk or diskette or on a basic-exchange diskette.

Create a Partitioned Data Set. This option creates a partitioned data set in a logical volume on the specified disk or diskette.

Create a Logical Volume. This option creates a logical volume on the specified extended-format disk or diskette.

Delete a Logical Volume, Data Set, or Member. This option removes the specified logical volume, data set, or member from the directory of the device, volume, or data set in which it resides. The space is therefore free and available for reallocation.

Delete Multiple Selected Logical Volumes, Data Sets, or Members. This option removes the specified logical volumes, data sets, or members from the appropriate directory.

Rename a Data Set, Member, or Logical Volume. This option replaces the name of a specified data set, member, or logical volume with a specified new name.

Rename a Volume. This option replaces the name of a specified volume with a specified new name.

Build a Communications Data Set Definition. This option creates a communications DSD in the specified task set library on the specified device.

Build a Direct-Access Data Set Definition. This option creates a DSD for a disk or diskette in the specified task set library on the specified device.

Build a Display Station Data Set Definition. This option creates a DSD for a display station in the specified task set library on the specified device.

Build an Operator Station Data Set Definition. This option creates a DSD for an operator station in the specified task set library on the specified device.

Build a Printer Data Set Definition. This option creates a printer DSD in the specified task set library on the specified device.

Build a Sensor I/O Data Set Definition. This option creates a sensor-I/O DSD in the specified task set library on the specified device.

Build a Timer Data Set Definition. This option creates a timer DSD in the specified task set library on the specified device.

Delete a Data Set Definition. This option deletes a DSD from a DSD table in the specified task set library.

Initialize

This utility formats a diskette as either basic-exchange format or extended format. (Basic-exchange format has 128-byte sectors, and extended format has either 256- or 512-byte sectors.)

The utility checks and flags defective cylinders, then initializes the volume label and header labels.

IPLmaint

Use this utility command to either:

- Prepare a diskette for IPL of the stand-alone processor-storage-to-diskette dump utility
- Change the name of the system to be loaded by the next IPL sequence, and specify whether the system to be loaded is the primary or alternate system

Merge

Use this utility command to combine two partitioned data sets or volumes into a third partitioned data set or volume. After the merge, you still have the two original data sets plus the target data set that contains the combined entries.

Patch

Use this utility to modify information stored on a disk or diskette device, volume, data set, or member. The input for this utility comes from either the operator station or from a disk or diskette data set or member.

This utility is different from the others, in that it prompts you at two times. Once you have described the source of the input records, and the devices, volume, data set, or member to be patched, you are prompted again. In your reply, you specify whether you want to patch the data or just display it.

If you choose to patch, you have a further option of verifying the current data before the patch is applied.

Report

This utility generates these reports:

- Directory (or table of contents)—This is a listing of a device, volume, or data set directory. The directory for the source device, volume or data set is formatted and written to the target device, volume, data set or member specified. You can select (1) a listing of all volumes on a disk or diskette, (2) a listing of all data sets within a volume, or (3) a listing of all members within a data set.
- Data Dump—This is a hexadecimal dump of data contained in a device, volume, data set, or member. The data from the source device, volume, data set, or member is formatted and written to the specified target device, volume or data set. The data dump starts at the location specified by the relative record and relative byte, and it continues for the number of records or bytes specified by the record count and byte count. If you do not specify these parameters, the dump continues until the end of data is reached. The output can be printed on a hard-copy device. Valid EBCDIC characters are also printed.
- Storage Image Dump—This listing is a previously acquired storage dump on a disk or diskette that is converted to printable form. The dump can be

generated by either the Series/1 abend processor or the stand-alone processor-storage-to-diskette-dump utility. You can produce a full or partial hexadecimal dump, which can be printed on a hard-copy device. Valid EBCDIC characters are also printed. In addition, abend information, registers, storage keys, and Series/1 control blocks can be printed.

Related Publication

IBM Series/1 Realtime Programming System: Operator Commands and Utilities

Chapter 6. Program Preparation

The Series/1 program preparation facilities consist of four program products that are separate from the operating system:

- The Program Preparation Subsystem (also referred to as *the subsystem*), which comprises:
  - The job stream processor
  - The text editor
  - The macro assembler
  - The application builder
- FORTRAN
- Mathematical and Functional Subroutine Library
- PL/I

These program products prepare the application programs that run under the operating system. In addition, each program product runs under the operating system in a batch environment.

The Job Stream Processor

The job stream processor is the part of the subsystem that controls the execution of the other programs in the subsystem, as well as your own batch programs. It has a control language for invoking programs and defining the data sets they use. The job stream processor reads the control statements (such as JOB, EXEC, PARM), analyzes the parameters you supply, and processes your requests for executing programs.

The job stream processor is your interface to the text editor, assembler, application builder, FORTRAN, PL/I, and your own batch programs. It manages the data sets and devices used by the programs and handles the automatic job-to-job, step-by-step transition during batch job input stream processing.

The Text Editor

The text editor is a part of the subsystem. With it, you can create, modify, list, and save text modules. You can use the editor in an interactive mode, entering text and commands from an operator station, or you can use it in a noninteractive mode by defining a disk or diskette data set as the command input source.

Using the editor, you can create a new text module or retrieve an existing text module and update it. The editor has 15 commands that have a variety of editing capabilities. The output from the text editor is a newly-created or updated text module that can be saved in a disk or diskette data set and later used as input to the macro assembler, FORTRAN, PL/I, or another program.

The Macro Assembler

The macro assembler translates macro, machine, and assembler instructions into one or more relocatable object modules and generates a program listing. Source statements are read in from a data set in the input stream or from a library data set.

The object modules produced by the assembler consist of object code and information that the application builder uses for building relocatable modules and resolving external symbols.

The assembler also produces various types of listings:

- Source statements and macro expansions
- External symbol dictionary and relocation dictionary
- Cross-reference table
- Error messages and statistics

You can control the types of listings the assembler generates by the options you specify.

The Application Builder

The application builder handles the final steps in program preparation—processing the object modules produced by the assembler or another language translator. The application builder does this processing in three phases:

Phase 1. This phase can create a load module in absolute format or a *composite module* in relocatable format. Absolute load modules are not executable under the realtime programming system. A composite module, in relocatable format, is used as input to phase 3 processing. A composite module is made up of object modules (programs) structured into a resident segment and optional overlay segments.

Phase 2. This phase builds control modules and prebind modules. The *control module*, which is required by the operating system, contains the tables and control blocks that the system needs to execute the functions requested by the task set (data management, queuing, and tasking, for example). The *prebind module*, which is optional, allows you to specify the information needed for binding the required resources to an application task set when it is installed rather than at execution time. This prebinding of resources allows a task set to start execution faster.

Phase 3. This phase creates task sets that are executable under the realtime programming system.

To create an application program that can execute in the realtime programming system environment, you must use phases 1, 2, and 3 of the application builder. Phases 1 and 2 create modules that are the input to phase 3, which combines them into a task set and writes that task set to a *task set library.* A task set library is a volume containing all the modules and tables associated with a single task set.

In phase 3, you must supply partition information. A task set must be prepared for a specific partition because it is not relocated when it is loaded.

A task set can have a simple structure, where all segments are resident during execution, or it can have a complex structure (overlay), where some segments reside on disk. In overlay structure, a storage-resident segment of code can request that the system retrieve a unit of code from disk and pass control to it. The specific structure and characteristics of the task set are defined by the information you supply on control statements as input to the application builder.

The printed output from the application builder is:

- Module map listings (for either composite modules or absolute load modules)
- Task set map listings
- Diagnostic messages and control statement listings

FORTRAN

FORTRAN is a high-level, mathematically oriented language designed to manipulate numeric data and format input/output operations. The language increases programming productivity because it requires less coding than assembler language. FORTRAN for the Series/1 consists of a compiler, a subroutine library, and an optional system support library.

The FORTRAN compiler automatically produces code with emphasis on compact storage and execution speed. With FORTRAN and your Series/1, you can:

- Perform calculations and make decisions
- Read information from sensor devices and send commands back to controlling devices
- Schedule programs and input/output operations
- Create reports and disk data sets

The FORTRAN subroutine library consists of routines for mathematical calculations, data conversion, error handling, and input/output operations. Only the library routines that you need are included in your program; your program need not contain any unnecessary code. Because each routine is reenterable, it is included only once in a storage load, resulting in further storage savings.

Series/1 also offers an optional FORTRAN system support library as a separate program product. This library includes process I/O, system service interface, time-of-day, and date subroutines.

Mathematical and Functional Subroutine Library (MFSL)

MFSL is a set of subroutines for application programming on the Series/1. It has mathematical, conversion, error-checking, and service subroutines that can be used by application programs written in Series/1 FORTRAN or assembler language.

PL/I

Series/1 PL/I is a high-level, general-purpose language. It can increase programming productivity in such areas as:

- Realtime applications
- Scientific and problem-solving applications
- Traditional data processing
- Transaction processing

Highlights of Series/1 PL/I are:

- Language extensions (ANS PL/I subset plus realtime and sensor I/O support)
- Extensive I/O capability
- Multiple data types and organizations
- Data manipulation features
- Extensive execution-time error-handling with a programmable error-handling capability (ON-handling language)
- Program modularity (PL/I block structure and scope rules)
- Storage efficiency (reenterable code, automatic storage allocation, and transient functions

The PL/I product consists of a compiler and a resident execution-support subroutine library, and a separately priced transient execution-support subroutine library.

Related Publications

For more information about program preparation, FORTRAN, MFSL and PL/I, refer to the following publications:

- *IBM Series/1 Program Preparation Subsystem: Introduction*, GC34-0121*
- *IBM Series/1 Program Preparation Subsystem: Batch User's Guide*
- *IBM Series/1 Program Preparation Subsystem: Text Editor User's Guide*
- *IBM Series/1 Program Preparation Subsystem: Macro Assembler User's Guide*, SC34-0124*
- *IBM Series/1 Program Preparation Subsystem: Macro Assembler Reference Summary*
- *IBM Series/1 Program Preparation Subsystem: Application Builder User's Guide*
- *IBM Series/1 Program Preparation Subsystem: Messages and Codes*
- *IBM Series/1 FORTRAN IV: Introduction* GC34-0132*
- *IBM Series/1 FORTRAN IV: Language Reference*, GC34-0133*
- *IBM Series/1 FORTRAN IV: User's Guide*
- *IBM Series/1 FORTRAN IV: Language Reference Card*
- *IBM Series/1 Mathematical and Functional Subroutine Library: Introduction*, GC34-0138*
- *IBM Series/1 Mathematical and Functional Subroutine Library: User's Guide*
- *IBM Series/1 PL/I: Introduction*, GC34-0084*
- *IBM Series/1 PL/I: Language Reference Manual*
- *IBM Series/1 PL/I: User's Guide*
- *IBM Series/1 PL/I: Messages*

*These publications are available now.

Appendix A. Summary of System Functions

The following chart lists the functions supported by the operating system and the various ways to request them from a program—through macros, PL/I, and FORTRAN.

System Function	*Macro*	*PL/I*	*FORTRAN*
Storage management			
Free dynamic storage in user partition	FREEMAIN	automatic	automatic
Free subpool segment	FREESEG		
Allocate dynamic storage in user partition	GETMAIN	automatic	automatic
Allocate subpool segment	GETSEG		
Define subpool	SUBPOOL		
Task set management			
Get completion code	GETCC		
Request task set identification	IDENTIFY	automatic	automatic
Terminate requesting task set and start new task set	LINKTS	TRANSFER TO statement	INVOKE statement
Queue task set to a partition	QUETS	RUN statement	CALL $TSQUE
Alter rollout/rollin status	SETROLL	CALL SETROLL	CALL $SETRL
Terminate a task set	STOPTS	STOP statement or STOP statement with ACTN option	CALL $TSSTP
Update system scheduler table	UPDSST	RUN statement with AT, AFTER EVERY, or SOURCE option, or UNSCHEDULE statement	CALL $MDSST, CALL START, or CALL TRNON
Update system task set table	UPDSTST	automatic	CALL $MDSTT
Task management			
Resume execution of a suspended task	RESUME	automatic	CALL START, CALL TRNON, or CALL WAIT
Create or start a new task	STARTASK	RUN statement	CALL $ATACH
Terminate a task	STOPTASK	STOP statement with ACTN option	CALL $DTACH
Suspend execution of a task	SUSPEND	automatic	CALL WAIT
Program management			
Request program execution	CALL	CALL statement	CALL statement
Define program entry	PROGRAM	PROCEDURE statement with TASK option	PROGRAM statement

System Function	*Macro*	*PL/I*	*FORTRAN*
Return from program (restore registers)	RETURN	RETURN or END statement	RETURN statement
Save registers and return address	SAVE	automatic	automatic
Event management			
Define an event	DEFEVENT	DECLARE statement with event attribute	CALL $DFNEV
Delete an event	DELEVENT	automatic	CALL $DLTEV
Post a completed event	POST	POST statement	CALL $POST
Reset an event	RESETEV	COMPLETION	CALL $name with event parameter
Wait for an event	WAIT	WAIT statement	CALL $AWAIT
Generate a list of events	WAITLIST	WAIT on multiple events	EXTERNAL; EVENT(s)
Queue management			
Define a queue	DEFQUE	OPEN statement with TRANSIENT attribute	CALL $DFNQU
Delete a queue	DELQUE	CLOSE statement with TRANSIENT attribute	CALL $DLTQU
Locate a queue element	RECEIVE	READ statement for transient file	CALL $DEQUE
Remove an element from a queue	REMOVE	READ statement for transient file	CALL $DEQUE
Add an element to a queue	SEND	WRITE statement for transient file	CALL $ENQUE
Timer management			
Convert time to different format	CNVTIME	automatic	automatic
Define a timer	DEFTIMR	DECLARE statement with AT attribute	automatic
Wait on a timer	DELAY	DELAY statement	CALL WAIT
Delete a timer	DELTIMR	automatic	automatic
Increase time of day	INCRTOD	CALL INCRTOD	automatic
Read time-of-day or date	READTOD	TIME, DATE, DAYNO built-in functions	CALL $RDTOD, CALL TIME, or CALL DATE
Change time-of-day or date	SETTOD	CALL SETTIME, CALL SETDATE, or CALL SETDAYNO	CALL $WRTOD
Stop a timer	STOPTIMR	DISCONNECT statement	CALL START CALL TRNON
Start a timer	STRTTIMR	WAIT, CONNECT, or RUN statement	CALL START CALL TRNON
Management of serially reusable resources			
Define a request control block	DEFRESC	automatic	CALL $DFNRS
Delete a request control block	DELRESC	automatic	CALL $DLTRS
Relinquish control of a logical resource	RELEASE	UNLOCK statement	CALL $RELRS
Provide serial access to a logical resource	REQUEST	LOCK statement	CALL $REQRS

System Function	*Macro*	*PL/I*	*FORTRAN*
Class interrupt management			
Read programmer-console data buffer	READFFC		
Write to the programmer-console data buffer	WRITEFFC		
I/O interrupt management			
Connect to a PI interrupt	CONNECT	CONNECT statement for PI type EVENT	CALL $CON
Disconnect from a PI interrupt	DISCONN	DISCONNECT statement for PI type EVENT	CALL $DISCN
Error management			
Check operator mode switch	CHKOMS		
Copy primary storage	DUMP	CALL DUMP	
Return error-exit status	ERRET	ON-handling	CALL $name with return code parameter
Format and route a message	MSG	Print execution-time messages	Print execution-time messages
Reset disk-resident data set or member	RESETEOD		
Reload and restart supervisor	RESTART	CALL RELOAD	
Retrieve record from error log	RETRIEVE		
Activate task error exit	SETERXIT	automatic	automatic
Write to error log	WTL		
Operator interface			
Write to operator station	WTC	DISPLAY statement	PAUSE
Write to operator station with reply	WTCR	DISPLAY statement with REPLY option	PAUSE with interrupt
Data management			
Build an immediate device control block	BLDIDCB	automatic	automatic
Build a device control block	BLDDCB	automatic	automatic
Execute I/O	EXIO	automatic	automatic
Data set management			
Build a data set definition	BLDDSD	ENVIRONMENT	
Build an I/O queue element	BLDIOQE		
Complete an asynchronous GET/PUT request	CHECK	WAIT statement	CALL WAIT
Close a data set	CLOSE	CLOSE statement	CLOSE statement
Close a prefound member	CLOSEM	ENVIRONMENT	
Request device-dependent functions	CONTROL	ENVIRONMENT	
Convert variables to and from EBCDIC	CONVERT	Conversion routine in PL/I library	Uses MFSL routines
End-of-volume processing	EOV		READ (,EOF=,ERR=)

System Function	*Macro*	*PL/I*	*FORTRAN*
Extract data management information	EXTRACT		
Get input from a device	GET	READ or GET statement	READ statement
Locate a PDS member or members	LOCATE	automatic	automatic
Relate the current (-1) data set position	NOTE		
Open a data set	OPEN	OPEN statement	automatic
Open a prefound member	OPENM	ENVIRONMENT	
Read input from a data set	READ	READ statement	READ statement
Reposition a data set	POINT	READ statement with IGNORE option	
Send a logical record to a data set or device	PUT	WRITE or PUT statement	WRITE statement
Set end of data	SETEODS		READ/WRITE with EOF
Translate a data area	TRANSLATE	automatic	automatic
Write output to a data set	WRITE	PUT, WRITE, or REWRITE statement	WRITE statement
Read analog input	READ	READ statement	CALL AIRDW or CALL AISQW
Write analog input	WRITE	WRITE statement	CALL AOW
Read digital input	READ	READ statement	CALL DIW
Write digital output	WRITE	WRITE statement	CALL DOLW or CALL DOMW
Communications			
Define switched-line answer list	SWLIST		

These functions are available to assembler-language programmers for writing code to execute in supervisor state and for performing operations not usually done in application programs.

Note. Macros marked with an asterisk are for use in supervisor state *only.*

System Function	*Macro*
Storage management	
Allocate storage from any partition	GETSTG*
Task set management	
Change I/O usage count	UPIOC*
Program management	
Execute system program	EXEC*
Return from system program	EXIT*
Load transient	LOAD
Generate parameter list	PARMLIST
Generate resolved parameter list	RESOLVE*
Remove transient from storage	UNLOAD
Event management	
Post device service task	DSTPOST*
Reset a device service task event	DSTRESET*
Device service task wait	DSTWAIT*
Management of serially reusable resources	
Request use of a gate control block	GATE*
Relinquish use of a gate control block	UNGATE*
I/O interrupt management	
Define an interrupt	DEFINT*
Delete an interrupt	DELINT*
Wait for an interrupt	INTWAIT*
Error management	
Define and activate an error log	DEFLOG
Delete and deactivate an error log	DELLOG
Record error status	LOG*
Set operator mode switch	SETOMS
Terminate system operation or system function	SYSTERM*
Operator interface	
Start a system command reader	STARTCR*
Stop a command reader	STOPCR*
Data management	
Display trace variables	TRCDIS
Inhibit I/O tracing	TRCINH
Trace I/O request	TRCIO
Resume I/O tracing	TRCRES
Start I/O tracing	TRCSTR
Stop I/O tracing	TRCSTOP

System Function	*Macro*
Device management	
Connect a logical device number to a physical device address	ASSIGN
Build a device descriptor block	BLDDDB
Build a device handler program list	BLDDHPL
Build a device I/O control block	BLDIOCB
Build a device handler program list control block	BLDPLCB
Remove a physical volume	DEMOUNT
Mount a physical volume	MOUNT
Start a device	STARTDEV
Stop a device	STOPDEV
Switch between a device and an alternate	SWITCH
System execute I/O	SYSIO*
Take a device offline	VARYOFF
Bring a device online	VARYON
Data set management	
Create a direct-access data set	CREATE
Define, change, or read data set definition	DEFDSD
Delete a DSD entry	DELDSD
Delete a direct-access data set	DELETE
Rename a volume, data set, or member	RENAME

Appendix B. Glossary

The following reference symbols are used in this glossary:

Contrast with. This refers to term that has an opposed or substantively different meaning.

Synonym for. This indicates that the term has the same meaning as another term, which is defined.

Synonymous with. This identifies terms that are synonyms for the term being defined.

See. This refers to multiple-word terms that have the same last word and are defined.

See also. This refers to related terms that have a similar, but not synonymous, meaning.

absolute task set. A task set which executes in a specific partition and has address constants adjusted according to the partition origin storage address. An output of the application builder.

ACB. Access control block.

access level. Three techniques available for accessing data: basic, logical, and physical. See also basic access level, logical access level, physical access level.

ACT. Abend control table.

active state. A software state in which a previously started task is in contention with other tasks in the system for control of the processor. There may be a maximum of four simultaneously active tasks (one per level); however, only one task at a time will have control of the processor.

addressing ID. The unique identifying character or characters associated with a communications station when writing to that station.

address space. The range of main storage (0 to 64KB) addressable by a single user.

allocated storage. See storage.

alternate device. A device assigned as a backup unit to another device. Requests may be manually switched from a device to its assigned alternate. Contrast with primary device.

answer list. A list of switched-line station IDs with any required control information. This list is used to insure that the switched line is connected to an authorized user.

application build. See application builder.

application builder. The program preparation facility (operating in conjunction with the job stream processor under control of the realtime programming system supervisor) that prepares the object module output of language translators for execution. To create output that is executable in a user-provided environment, it can be used to create an absolute load module (an output of phase 1 processing). To create output that is executable under the realtime programming system, it can be used to create a task set (the final output from processing performed in phases 1, 2, and 3).

asynchronous data transfer. A physical transfer of data to or from a device that occurs without a regular or predictable time relationship following the execution of an I/O request. Contrast with synchronous data transfer.

asynchronous timer. A logical timer which runs concurrently with the task that caused its creation. Asnychronous timers are normally used for timeouts.

attended mode. A system operating mode indicating that an operator is present at the system and a system console is included at the system.

auto-call library. Disk resident load module library used by the application builder to obtain composite modules that contain a program that can resolve a reference from another program in a task set.

auxiliary storage. See storage.

basic access level. A supervisor interface that provides access to data on a device by physical record through execution of requests (EXIO) for hardware I/O operations.

batch execution. Program execution initiated by the job stream processor in response to job control statements.

batch partition. The user partition that is being used by and managed by the job stream processor.

batch program. Any program preparation facility or user task set running under the control of the job stream processor.

BCB. Buffer control block.

binding. The assignment or connection of resources or objects to a task set. This occurs before task set execution.

block. The portion of a data set accessed at the physical level (READ/WRITE). A block may contain one or more physical records and may be contained in one or more physical records.

bound task set load module. A task set load module which has been bound to its execution environment. The bound task set load module contains the image of the partition at the completion of task set installation. See unbound task set load module, task set load module.

buffered device. A device which has I/O elements queued to a direct access device before being written.

CCCB. Completion code control block.

CDC. TP device control block.

CDT. Command definition table.

CIDC. Close interrupt control table.

class interrupt. A hardware interrupt from an SVC instruction, program check, full-function console, trace, power/terminal check, or machine check.

CM. Control module.

command. A character string which represents a request for action within the system from a source external to the system.

command processor. A system task which processes a set of commands from a queue.

command reader. A system task which reads commands from an input data set and routes each command to the command processor. A maximum of two command readers may be active at the same time.

common control section. A type of control section that reserves an area of storage. It can be referred to by resident and overlay segments within a task set.

composite module. Object modules (programs) structured into a resident segment and optional overlay segments. A composite module is in relocatable format, that is, its address constants can be modified to compensate for a change in its origin.

consecutive data set. A collection of data having a consecutive arrangement to which the system has access.

consecutive data set organization. Blocks and records are physically consecutive from the beginning of the data set to the end-of-data indicator. Data is processed by accessing the block or record which follows the last block or record accessed. End-of-data need not coincide with the end of the data set. All data sets on serial devices have consecutive organization. Data sets on direct access devices may have consecutive organization.

consecutive organization. See consecutive data set organization.

control module (CM). A set of tables and control blocks that contain control and parameter information pertaining to the task set. It is one of the modules produced by the application builder and subsequently included in the task set load module.

cycle steal. A hardware condition that occurs when an I/O device shares processor storage cycles with the processor.

data set. A named collection of data which resides on a device. See consecutive data set, direct data set, partitioned data set.

data set control block (DSCB). Each level of a data set is described by a DSCB while the data set is open.

data set definition (DSD). Describes and locates a data set being used by a task set. The DSD exists in the using program or in a DSD table data set in the task set library. The DSD is accessed when the data set is opened.

data set definition name (DSD name). The external name of a DSD table entry used within a task set to reference the data set described by that entry.

data set definition statement (DSD statement). A job stream processor control statement that allows the user to establish a connection between a data set or device and a DSD name used in a program.

data set definition table (DSD table). A table that contains parameters for data sets.

data set name (DS name). The term or phrase used to identify a data set. It is contained in the data set definition table of each task set referencing that data set.

DCB. Device control block.

DCT. Dispatcher control table.

DDB. Device descriptor block.

device. A piece of mechanical, electrical, or electronic equipment used to contain data that is input to or output from the processor.

device address. Physical device address recognized by the hardware.

device backup. Pertaining to the assignment of alternate devices. See manual device backup.

device dependent program. A program that must consider the characteristics of a specific type of I/O device when processing an I/O request.

device descriptor module (DDM). A program unit containing needed information required to manage a device by a device service task. Device descriptors are located in the resident load module or in a data set in the task set library of the system task set.

device handler. A combination of a device service task and an interrupt service task.

device-independent program. A program that does not consider the characteristics of a specific type of input/output device when processing an input/output request.

device line. The actual number of characters that can be displayed on one print line of the device. For example, the operator station line length is 72 characters, the display station line length is 80 characters.

device name. The logical name assigned to a device.

device number. The logical device identifier used by device service tasks to reference a device address.

device service task. A queue-driven task that processes requests for data transfer and performs the I/O operations necessary to effect the requested functions on a specific device.

DICB. Data integrity control block.

DIOCB. Device I/O control block.

direct access device. See direct access storage device (DASD).

direct data set. A data set whose records are in random order on a direct access volume. Each record is stored or retrieved according to its actual address or its address relative to the beginning of the data set.

direct organization. The organization of records in a data set created by the direct access method.

disk queue. A queue which has its elements on a direct access device. Contrast with storage queue.

dispatch. The act of allocating the processor to a task so that it can execute instructions on behalf of the task.

DQCB. Disk queue control block.

DSD. Data set descriptor.

DSD environment. See environment.

DSD name. See data set definition name.

DSD statement. See data set definition statement.

DSDT or DSD table. See data set definition table.

DVT. Device vector table.

dynamic storage. (1) A device storing data in a manner that permits the data to move or vary with such that the specified data are not always available for recovery. (2) The available storage left within the partition after the task set is loaded.

ECB. Event control block. A control block used to manage events.

editing session. A period of time beginning when the editor is invoked and ending when the editor has completed processing.

end-of-data indicator. A code which signals that the last record of a consecutive data set has been read.

environment (of a data set). The data set definitions that are in effect at any point in time during a batch session.

environment list. A data set (or member) containing a group of data set definitions which comprise an environment.

ERB. Execution request block. A control block that contains registers and data for execution of a supervisor program.

external name. The symbolic EBCDIC name for a resource.

first level interrupt handler (FLIH). A system-provided subroutine resident in the system task set. A FLIH fields a particular type hardware interrupt.

fixed-line number. The line number assigned to a text record and associated with that text record for the duration of the editing work session (unless specifically altered by the user).

fixed partition. A partition having a predefined beginning and ending storage address.

FLIH. First level interrupt handler.

free storage. See storage.

FSCB. Free storage control block.

gate. (a) A request for access to a serially reusable object. (b) To control access to a serially reusable resource.

GCB. Gate control block. A control block used to manage supervisor requests for a serially reusable resource.

generation input stream. An input stream created by the generation program which, when executed, produces a system tailored to the user responses to system generation questions.

global area. An uninitialized portion of a partition accessible by any program of a task set in the partition at a given time. The same area may be used by other task sets that execute in the same partition. The size of the GLOBAL area is determined by the collective sizes for the largest uniquely named (or unnamed) GLOBAL section definitions. These definitions are declared by programs that make up a task set.

global control section. A type of control section that reserves an area of storage. It can be referred to by any primary or secondary program and their associated overlays within a task set. See also global area.

global object. An object, such as a queue, which may be shared by all task sets in the system.

global parameter buffer. A buffer in the global portion of the partition, accessible to batch programs, where the job stream processor stores the PARM statement parameter string.

GPR. General purpose register.

ICB. Interrupt control block.

ICDS. Input command data set.

IDCB. Immediate device control block.

input command data set (ICDS). A consecutive data set from which a command reader or command processor may obtain commands or data.

interactive. A realtime interface between a user and a program system.

internal name. A 16-bit binary value assigned as a result of an NCON, which is used to represent an EBCDIC external name within the system. See name constant.

interrupt service task (IST). A user or system-supplied task which is connected to an I/O interrupt and is dispatched by a first level interrupt handler (FLIH) on occurrence of the interrupt.

IOQE. I/O queue element.

IST. Interrupt service task.

job. A collection of related problem programs, identified in the input stream by a JOB statement followed by one or more EXEC and data set definition statements.

job stream. See input stream.

job stream processor. A component that reads and interprets job control statements and satisfies requests made by those statements.

KB. Kilobytes.

LCT. Log control table.

LDT. Logical device table.

level of access. See access level.

line. A string of characters accepted by the system as a single block of input from an operator station; for example, all characters entered before the carriage return key or the ENTER key is pressed. For the text editor, it represents the line number plus the text line.

line display range. That portion of a line to be displayed when the text editor lists a line.

line length. Logical record length of lines being edited by the text editor.

logical data transfer. The transfer of a logical record to or from a buffer which may or may not cause one or more asynchronous I/O data transfers to take place.

logical resource. An entity which represents the use of physical resources for a particular function, for example, a program, event, queue, or data set. This entity is assigned, and referred to by, an object name.

logical timer. A software logic element representing the usage of a hardware timer.

logical volume. A portion of a physical volume which is viewed by the system as a volume. See volume.

LPB. Load program block.

manual device backup. A feature that allows the operator to switch subsequent

I/O request to an alternate device or to switch from the alternate device back to the primary device.

message buffering facility. Facility which allows messages to overflow and backup into a disk data set; the messages are written as the I/O queue empties.

MIOCB. Master I/O control block.

MTRCB. Master timer control block.

multithreading. Pertaining to the concurrent operation of more than one path of execution within a computer.

name constant. A data type which specifies that the variable is the internal name of an object. Synonymous with NCON.

NCON. See name constant and internal name.

non-interactive. An indirect interface between a user and a program system. (For example, through a disk or diskette data set.)

non-storage device. A device not having the ability to retain data transferred to or from it for subsequent retrieval.

non-switched point-to-point line. A single communications line that contains one permanently connected communications station.

nucleus. (1) A task set load module for the system task set that contains machine interface data, the system communication table, master control blocks, SVC routines, and system tasks and programs, and is write-protected from user task sets. (2) The task set load module that is brought into main storage by the bootstrap loader following IPL.

null record. A record containing a null character string used to format space on the direct access device when a direct data set is created.

object. A logical resource which is managed by the supervisor. Each object is assigned a name so that it can be referenced by both the user and the supervisor. It may imply the use of one or more physical resources.

object code compability. (1) Pertaining to a system where changes to the system do not require recompilation or assembly of user programs. (2) Contrast with source code compatibility.

object definition. (1) The set of information required to create and manage an object. (2) The creation of a control block for an object. It also defines the object as available to the user.

object module. The output of a single assembly or compilation containing one or more control sections—CSECTs. An object module is equivalent to a program.

object module data sets. Disk resident data sets that contain object modules.

object program. That part of an object module that constitutes a control section. It also is the output of the macro assembler.

OCDS. Output command data set.

OEM. Original equipment manufacturer.

operator station. The device used for primary interactions between the user and the software. It can be either a teletypewriter or a display station.

OPICT. Operator interface control block.

output command data set (OCDS). A consecutive data set to which a command reader or command processor may write information concerning the processing of commands or input data.

overlay. (a) A segment of a program that is not permanently located in storage during task set execution. (b) The technique of repeatedly using the same areas of a task set during execution of a program.

overlay area. An area within the task set load module of a task set used for the execution of overlay segmentss which are not permanently located in storage. An overlay area is associated with a resident segment during application build.

overlay module. A structure that contains all overlay segments in a single task set. It is one of the modules produced by the application builder as part of a task set library.

overlay segment. A segment that resides on secondary storage and is loaded into the overlay area associated with its resident segment. All overlay segments in a composite module are assoicated with the resident segment in that composite module. A program in an overlay segment can call a program within the same overlay segment or a program in any resident segment.

partition. A segment of physical and addressable storage which may contain one task set at a time. A partition begins and ends on a 2KB boundary and has a unique numeric ID from 0 to 15. See also fixed partition; system partition; user partition.

partitioned data set. A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

PCD. Partition control descriptor.

PCT. Partition control table.

permanent object. An object which is defined to the supervisor by an explicit request. The object's control block is not deallocated until deletion is explicitly requested or the task or task set terminates.

PHB. Program header block.

physical access level. (1) A supervisor interface that transfers one or more physical records, called a block, to or from a device, through execution of WRITE or READ functions. A block may contain all or part of one or more logical records. (2) Access to a data set by block, which may consist of one or more physical records.

physical record. The portion of a data set transferred at the basic level of access (EXIO). The size of a physical record is determined by the hardware device unless the device supports variable-length records, in which case, the physical record size equals the block size of the data set using the device.

physical resource. Any facility of the computer available to do work, such as, the CPU, main storage, or an I/O device.

physical timer. Synonym for hardware timer.

PLCB. Program list control block.

PMCT. Program management control table.

polling ID. The unique character or characters associated with a particular station when reading from that station.

PQEL. Partition queue element.

prebinding. The connection of task sets, control blocks and resources before task set execution. See also binding.

prebind module. A module that contains the specifications used during task set installation to create the bound task set load module. It is one of the modules produced by the application builder as part of a task set library.

primary device. Pertaining to a device that has an alternate device assigned to it. Contrast with alternate device.

primary program. The first program executed under a task. A primary program has a single entry point and must have a program header. A primary program is either reenterable, serially-reusable, or nonreusable and is only invoked by a start task request. Contrast with secondary program.

primary segment. The resident segment that contains the initial entry point of a task set. The entry point of a task set must be a primary program within the primary segment.

primary storage. See storage.

primary task. The first task activated when a task set begins execution. The primary task is started by the supervisor.

priority. Priority consists of the hardware level and the software priority within the level.

problem program state. See problem state.

processor. (1) In hardware, the resource required to execute an instruction stream. (2) In software, a synonym for processing program.

processor storage. The storage provided by one or more processing units. This term pertains to physical locations in hardware devices.

program. (1) a named sequence of instructions that operates under the auspices of a task. (2) A program is the output of a single assembly or compilation containing one or more control sections—CSECTS. A program is equivalent to an object module.

program management. That portion of the supervisor which manages requests for program execution.

program preparation facilities. The components used to create user task sets. Specifically, the text editor, macro assembler, and application builder.

protected free storage. See storage.

QCB. Queue control block. A control block used to manage queues.

QECB. Queue element control block.

question file. A file containing information which may be used to produce question forms for batch input or to allow questions to be asked interactively. The questions are organized in a series of specification levels. Response to higher specification levels may generate or ignore questions at lower specification levels.

queue data set. A data set on a direct access device used to contain one or more queues.

queue-driven task. A task whose unit of work is represented by an element in a queue.

queue element. (1) A block of data in a queue. (2) One item in a queue.

random data set. See direct data set.

RCB. Request control block. A control block used to manage user requests for a serially reusable resource.

ready state. Software state in which the task is ready to be activated and is contending for processor execution time.

record. The portion of a data set accessed at the logical level (GET/PUT).

refreshable program. A program that can at any time be replaced with a new copy without changing either the sequence or results of processing.

relative block number. A number that identifies the location of a block expressed as a difference with respect to a base address. The relative block number is used to retrieve that block from the data set.

relative-record number. A number that indicates the location of a logical record, expressed as a difference with respect to a base address. The relative record number is used to retrieve that logical record from the data set.

resident. Pertaining to that portion of a task set which is permanently located in storage for the duration of the task set execution.

resident load module (RLM). A module within the task set load module. It is made up of the programs of a task set which remain resident in storage for the duration of the task set. See also task set load module.

resident segment. A segment that remains in primary storage for the duration of task set execution. A program in a resident segment may call a program in one of its overlay segments or a program in another resident segment.

return code. A code used to influence the execution of succeeding steps in a job in the input stream. An indicator, which is passed from a batch program to the job stream processor, that reflects the status of the batch program at the time of its termination.

RLD. Relocation dictionary.

roll in. Restoring a partition to the task set that was previously rolled out. Roll in occurs when the rolled out task set has the highest queue priority in the partition queue. See also rollout/rollin.

roll out. To transfer to task set out of a partition and replace the task set with a task set which has a higher queue priority. No subsequent roll out wili occur until a roll in restores the partition. The shared task set may not be rolled out. See also rollout/rollin.

rollout/rollin. An optional facility specified at system generation that allows the temporary reassignment of a partition from one task set to another of a higher priority. The contents of the partition is placed on a disk storage device. A single fixed partition is designed as the rollout/rollin partition at system generation or IPL time. See also roll in; roll out.

root segment. A segment of a program that is permanently located in storage for the duration of task set execution. See also resident segment.

RQE. Reply queue element.

scheduler task. A queue-driven task in the system task set which manages the scheduling of task sets.

scheduling. The ability to request that a task set should be started at a particular time of day or after a specified time interval or on occurrence of a specified PI interrupt.

SCT. System communication table.

secondary program. Any program other than the primary program of a task. A secondary program may have multiple entry points and may or may not have a program header. Secondary programs are invoked by a call request or by direct linkages, such as, assembler branch instructions. See also primary program.

secondary segment. Any resident segment other than the primary segment of a task set.

secondary task. Any task other than the primary task in a task set. A secondary task is started by a program executing under either the primary or another secondary task.

segment. A structure containing one or more programs, which is a portion of a composite module or a task set load module.

sequential access method. An access method where: (a) Data is processed by accessing the block or record previously accessed. Either consecutive or random data sets may be accessed using the sequential access method. The user establishes the order of the records when he creates the data set. (b) Blocks and records are accessed sequentially. Access can begin at any record in the data set and can be forward (until the end of data is reached) or backward (until the beginning of the data set is reached).

sequential organization. Synonym for consecutive organization.

serially reusable resource (SRR). A logical resource or an object which can be accessed by one task at a time.

shared task set. A special type of user task set that contains tasks, programs, event definitions, queues, timers, and data sets to be shared among user task sets.

SICB. Sub-interrupt control block.

SLIH. Second level interrupt handler.

source code compatibility. (1) Pertaining to a system where changes to the system do not require changes to a user's source code, but may require a recompile or assembly of the source code. (2) Contrast with object code compatibility.

source module. A collection of source statements which constitute the input to a language translator for a particular translation. These source statements may be created, modified, and listed using the text editor.

spanned record. A record that is contained in more than one block.

split screen. The division into sections of a display screen in a manner which allows two or more programs to use the display screen concurrently.

SRR. Serially reusable resource.

SSA. Status save area.

SST. System scheduler table.

starter system. A system task set (supervisor) in IPL format which supports the system generation process.

step. A request to the job stream processor to execute a program and, optionally, to define data sets used by the program. Related programs are commonly executed by a series of steps within a job.

storage. The first 64KB of processor storage is known as primary storage.

Secondary storage is processor storage beyond the first 64KB. Primary storage consists of allocated storage (allocated to a partition) and unused primary storage (not allocated to a partition). Free primary storage is storage, in a partition that is not allocated to a task set. This type of storage is dynamically allocated by the supervisor in response to system or user requests. Free primary storage consists of protected free storage and unprotected free storage. Protected free storage is for system use, such as control blocks and save areas. Unprotected free storage is available to the user or the supervisor for any requests.

storage queue. (1) A queue which has all its elements resident in main storage. (2) Contrast with disk queue.

STST. System task set table.

subprocessor. A program preparation facility or a user program.

subroutine. A sequence of instructions which is internal to a program and is not known to, or controlled by, the supervisor.

supervisor services. A general term for all functions in the supervisor that are available to the user through a supervisor interface.

suspended state. A software state in which the task will not be dispatched by the system and is not contending for the processor. In Series/1, if a suspended task is resumed, it returns to the state it had prior to being suspended.

SVT. System variable table.

switched line. A single communications facility with only one station. The station may be disconnected when the facility is not in use. While connected, this facility is identical to a point-to-point line.

symbolic priority. See priority.

synchronous data transfer. A physical transfer of data to or from a device that has a predictable time relationship with the execution of an I/O request.

synchronous timer. A logical timer used to place the task that created it in the wait state for a specified time interval. Synchronous timers are used for delay operations.

SYSGEN. System generation.

System generation. The processing of options to create a supervisor tailored to the user's needs.

system nucleus. See nucleus.

system partition. A partition containing the system task set. Its partition number is 0.

system services. A general term for all services made available to the user by the system, including supervisor services.

system task set. (1) A task set that provides system services. (2) That portion of a supervisor which may be totally or partially resident. (3) A collection of tasks, programs, and data sets required for the supervisor. See also task set.

system timer task. In the realtime programming system, that portion of the supervisor that manages the hardware timers used for timer services and/or time of day services.

table of contents (TOC). The directory of a permanent data set which describes and locates the subsidiary data sets contained within that data set.

TACT. Transient area control table. A table used to manage transient areas.

TAD. Transient area descriptor. A control block that describes a transient area and its contents.

task. The dispatchable entity used by the supervisor to establish and track concurrent program execution within the system. Each task represents a single thread of execution through a program or set of programs. The first program executed under each task is a primary program. All others are secondary programs.

task error exit routine. An optional user-written program which is given control when the task abnormally terminates.

task set. A named collection of programs, data, and control blocks designed to execute within a partition. The programs of a task set perform a related set of work and execute under one or more tasks.

task set library. A logical volume containing all of the data sets associated with a single task set. A task set library may also contain user data sets.

task set load module. A structure that contains all resident segments, their associated common and overlay areas, and the control module, for a single task set. A task set load module is loaded from a consecutive data set in the task set library into a partition when the request for the task set becomes the highest priority element in the partition queue. It remains in primary storage for the duration of task set execution. A task set load module is in absolute format, that is, its origin cannot be changed. See bound task set load module, unbound task set load module.

task start. The creation of a new task in the system.

task states. The states of execution status of a task relative to the processor; active state, ready state, suspended state, and wait state.

task switch. (1) Allocation of the processor to another task, for example, a ready or active task of higher priority than the current task in execution. (2) A change in the task that is in control of the processor. The new task's state changes from ready to active and the current task is placed in a state other than active.

TCB. Task control block.

text compression. An attribute of a permanent data set which specifies that strings of the same character are stored in a compressed form to reduce the space required for the data set.

text editor. The program preparation facility that is used to create, modify, and list text modules. Text prepared using the text editor may be in the form of source modules, which may be input to the macro assembler, *or* text data, which may be input to a user program or one of the program preparation subsystem programs.

text module. A term used by the text editor to indicate the data (text) that may be created and maintained using the facilities of the text editor. This data is usually in the form of printable characters (for example, source modules, data input to a user program).

timer. A mechanism for defining an interval of time. See asychronous timer, logical timer, physical timer, synchronous timer.

TOC. Table of contents.

TPT. Transient program table. A table that contains the disk address of transient programs.

transient area. A main storage area used for temporary storage of transient programs.

transient program. Self-relocating, refreshable program permanently stored in the task set library and loaded into a transient area or into dynamic storage when needed for execution.

TRE. Timer request element.

TSCB. Task set control block.

TWSD. Task work stack descriptor.

unattended mode. No operator present at system or no operator station included at system generation.

unbound task set load module. A task set load module which has not been bound to its execution environment. It is one of the modules produced by the application builder as part of a task set library. The unbound task set load module may be loaded into a partition without being prebound or it may be used as input to task set installation to create a bound task set load module. See bound task set load module, task set load module.

use count. A dynamic count of users kept for objects which may be used by multiple tasks.

user partition. A partition that contains a user task sets when in execution. The partition number can be from 1 to 15.

user task set. A collection of tasks, programs, and data sets required to perform a related set of operations for a user application. This task set executes (in problem program state) in a user partition. See also task set.

vary offline. (1) To change the status of a device from online to offline. When a device is offline, no data set may be opened on that device. (2) To place a device in a state where it is not available for use by the system; however, it is still available for executing I/O.

vary online. To restore a device to a state where it is available for use by the system.

VCBA. Variable control block area. Protected storage within the partition used by the supervisor for variable-length control blocks. It exists within the control module.

VDS. Volume control block/data set control block.

volume. (1) that portion of a single unit of storage which is accessible to a single read/write mechanism, for example, a drum, a disk pack, or part of a disk storage module. (2) A recording medium that is mounted and demounted as a unit, for example, a reel of magnetic tape, a disk pack, a data coil. (3) See logical volume.

work data sets. The data sets and data set members used by the program preparation facilities as temporary work areas.

work files. A data set or member used by the program preparation facilities as a temporary work area.

work stack. (1) A list that is constructed and maintained so that the next information to be retrieved is the most recently stored information in the list, that is, a last-in-first-out (LIFO) or pushdown list. (2) An area of unprotected main storage allocated to each task and used by the programs executed by that task.

wrap count. The number of times that the auto poll facility cycles through a polling list before it returns an end-of-list condition to the user.

write protection. A feature which prevents storage from being modified by a program running in problem state. Protected storage may be modified by a task running in supervisor state or by a cycle steal operation.

zero correction. See zero offset.

zero offset. An option of analog input. This option automatically compensates analog input to the ambient temperature.

Appendix C. Debugging Aids

Debugging aid:	*What it does:*
Display storage (DISP) operator command	Displays small areas of processor storage at the operator station.
Patch storage (PTCH) operator command	Patches small areas of processor storage through the operator station.
Report utilitiy	Dumps storage, data sets, or directories in formatted reports.
Hardware saving the level status block (LSB)	When a class interrupt occurs, the hardware saves the LSB. The operating system chains the LSB as an execution request block off the interrupted task control block.
Patch utility	Patches data within data sets.
EXTRACT macro	Extracts data management information from a data set's control blocks.
Trace I/O (TRCE) operator command	Traces all I/O operations on both communications and noncommunications devices.
Online terminal testing	A background program is invoked to test the active 2740 Model 1 terminal.
Log define and log delete (LOG) operator commands	All I/O errors and class interrupts can be recorded in a system error log through an operator command. You can write error records into a user error log through an operator command by setting up an error-exit routine.
RETRIEVE macro	Error records can be examined through either the RETRIEVE macro or the REPORT utility.
Stand-alone processor-storage-to-diskette dump	Dumps processor storage to a diskette.

For detailed information about debugging aids, refer to the *IBM Series/1 Realtime Programming System: Control Blocks and Debugging Guide.*

Appendix D. System-Generation Options

This appendix is a list of the options available to you during system generation. For a detailed explanation of the options and the system-generation procedures, refer to the *IBM Series/1 Realtime Programming System: Generation and Installation Procedures.*

System-generation options fall into five categories:

- IPL options
- Initialization commands
- Processor options
- Configuration options
- Service aids and error-handling options

IPL Options

Partition Definition. With this option, you specify:

- The size of the system partition
- For each user partition, the partition number, size, maximum number of entries in the partition queue, and, optionally, partition origin
- The rollout/rollin partition, if required

Automatic Device Start. With this option, you specify which I/O devices are to be started automatically at IPL time.

Initialization Commands

With this option, you specify the system commands and user commands that are to perform initialization activities such as starting task sets and defining logs.

Processor Options

Programmer Console. With this option, you specify whether your machine is equipped with a programmer console.

Timers. In this option, you specify whether hardware timers are part of the I/O configuration. If time-of-day support is required, you must specify the software support, as well as the precision for time-of-day and the format for the date.

Configuration Options

I/O Configuring. You must identify the devices used by the system, by defining their device addresses and device names.

Operator Station. With this option, you specify the I/O device to be used as the operator station.

Transient/Resident Selection. With this option, you can specify which supervisor programs are to be transient and which ones are to be storage-resident.

User-Supplied Supervisor Programs. Through this option, you specify whether you want to include your own supervisor programs.

Rollout/Rollin. You must specify whether you need rollout/rollin. If so, you must also define a rollout/rollin data set.

Scheduler. With this option, you specify whether you wish to schedule task sets.

Task Set Binding. You can specify the maximum number of task set libraries in the system task set table.

System Control Blocks. You must specify the number of execution request blocks required for operation of the generated system.

Floating Point. You must specify whether you wish to use the floating-point emulator.

System Command Selection. With this option, you can select the system commands to be associated with the system command processor.

ABEND Dump Option. In this option, you specify whether you want ABEND dump included in the system.

Online Terminal Testing. With this option, you can invoke a background program to test the active 2740 Model 1 Terminal.

Service Aids and Error-Handling Options

Battery Backup. This option lets you identify battery backup in case of a power failure.

Error Logging. With this option, you specify whether you want a system error log and user error logs. In addition, you must specify the number of error logs, their sizes, and where they are to reside (in storage or on disk).

Exit Routines. This option lets you specify your own exit routines for class interrupts and abnormal termination. In addition, you can supply a routine to set the time-of-day and date during IPL.

System Dump Data Set. You must specify the data-set-definition name of a data set to be used for storage dump.

I/O Trace. With this option, you must specify whether your system is to have I/O tracing.

Task Error Exit. Through this option, you specify whether you want to include task-error-exit subprograms that you have written.

Patch/Display. With this option, you specify whether you will want to display storage (up to 56 bytes) and patch storage (up to 20 bytes).

System Reload/Restart. Through this option, you specify whether you will want the system reloaded and restarted after a serious error has made the current system unable to continue.

Index

Realtime Programming System:
Introduction and Planning Guide
GC34-0102-0

READER'S COMMENT FORM

YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

Cut or Fold Along Line

What is your occupation?______________________

Number of latest Technical Newsletter (if any) concerning this publication: ______________

Please indicate your name and address in the space below if you wish a reply.

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

GC34-0102-0

Your comments, please . . .

This manual is part of a library that serves as a reference source for IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut Along Line

Fold

Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

IBM Corporation
Systems Publications, Dept 27T
P.O. Box 1328
Boca Raton, Florida 33432

Fold

Fold

IBM Series/1 Realtime Programming System: Introduction and Planning Guide Printed in U.S.A. GC34-0102-0

IBM

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
P.O. Box 2150, Atlanta, Georgia 30301
(U.S.A. only)

Realtime Programming System:
Introduction and Planning Guide
GC34-0102-0

READER'S COMMENT FORM

Cut or Fold Along Line

YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

What is your occupation?____

Number of latest Technical Newsletter (if any) concerning this publication: ________________

Please indicate your name and address in the space below if you wish a reply.

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Your comments, please . . .

This manual is part of a library that serves as a reference source for IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut Along Line

Fold Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

IBM Corporation
Systems Publications, Dept 27T
P.O. Box 1328
Boca Raton, Florida 33432

Fold Fold

IBM

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
P.O. Box 2150, Atlanta, Georgia 30301
(U.S.A. only)

International Business Machines Corporation

General Systems Division
5775D Glenridge Drive N.E.
P. O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

IBM Series/1 Realtime Programming System: Introduction and Planning Guide Printed in U.S.A. GC34-0102-0