GC21-7729-0

File No. S38-36

IBM System/38

IBM System/38
Control Program Facility Concepts Manual

Program Number 5714-SS1

First Edition (October 1978)

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, be sure you have the latest edition and any technical newsletters.

This publication is for planning purposes only. The information herein is subject to change before the products described become available.

Use this publication only for the purposes stated in the *Preface*.

Publications are not stocked at the address below. Requests for copies of IBM publications and for technical information about the system should be made to your IBM representative or to the branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatsoever. You may, of course, continue to use the information you supply.

Preface

This publication describes the concepts of the System/38 Control Program Facility. These concepts must be understood before decisions can be made about the overall design and use of a System/38 installation with the Control Program Facility.

This publication is intended for persons who are responsible for designing and maintaining a system installation, for programmers who write applications to be used on the system, and for anyone else who needs a general understanding of the functions provided by the System/38 Control Program Facility.

The chapters in this manual are designed to be read in sequence. The chapters present:

- An overview of the control program facility
- The manner in which the control program facility manages objects stored in the system
- The manner in which the control program facility manages work performed on the system
- The manner in which data is managed on the system
- The facilities provided to assist in application development on the system
- The facilities provided to assist in managing the operation of the entire system

This publication does not describe how to perform operations nor does it describe individual commands. Instead, the publication explains the concepts that must be understood before the Control Program Facility can be used efficiently.

Notes:

1. This publication follows the convention that *he* means *he or she*.
2. Some of the publications listed below can be ordered now; others will be available later. Where the name of the publication is followed by an order number, that publication can be ordered now.

Prerequisite Publications

- *IBM System/38 System Introduction*, GC21-7728. This publication summarizes what the System/38 is and how it can be used to meet an organization's application processing requirements.

Related Publications

- *IBM System/38 Guide to Publications*, GC21-7726. This publication contains the titles and reading sequence of related publications and describes the contents of each.
- *IBM System/38 Control Program Facility Reference Manual—Control Language*. This publication explains the control language syntax.
- *IBM System/38 Control Program Facility Reference Manual—Data Description Specifications*. This publication explains the specifications form used for describing data.
- *IBM System/38 Control Program Facility Programmer's Guide*. This publication describes how to use the functions introduced in the concepts manual and how to apply the commands and specifications explained in the reference manuals.
- *IBM System/38 Master Index*. This publication combines the indexes of the frequently used System/38 publications.

Preface

This publication describes the concepts of the System/38 Control Program Facility. These concepts must be understood before decisions can be made about the overall design and use of a System/38 installation with the Control Program Facility.

This publication is intended for persons who are responsible for designing and maintaining a system installation, for programmers who write applications to be used on the system, and for anyone else who needs a general understanding of the functions provided by the System/38 Control Program Facility.

The chapters in this manual are designed to be read in sequence. The chapters present:

- An overview of the control program facility
- The manner in which the control program facility manages objects stored in the system
- The manner in which the control program facility manages work performed on the system
- The manner in which data is managed on the system
- The facilities provided to assist in application development on the system
- The facilities provided to assist in managing the operation of the entire system

This publication does not describe how to perform operations nor does it describe individual commands. Instead, the publication explains the concepts that must be understood before the Control Program Facility can be used efficiently.

Notes:

1. This publication follows the convention that *he* means *he or she.*
2. Some of the publications listed below can be ordered now; others will be available later. Where the name of the publication is followed by an order number, that publication can be ordered now.

Prerequisite Publications

- *IBM System/38 System Introduction,* GC21-7728. This publication summarizes what the System/38 is and how it can be used to meet an organization's application processing requirements.

Related Publications

- *IBM System/38 Guide to Publications,* GC21-7726. This publication contains the titles and reading sequence of related publications and describes the contents of each.
- *IBM System/38 Control Program Facility Reference Manual—Control Language.* This publication explains the control language syntax.
- *IBM System/38 Control Program Facility Reference Manual—Data Description Specifications.* This publication explains the specifications form used for describing data.
- *IBM System/38 Control Program Facility Programmer's Guide.* This publication describes how to use the functions introduced in the concepts manual and how to apply the commands and specifications explained in the reference manuals.
- *IBM System/38 Master Index.* This publication combines the indexes of the frequently used System/38 publications.

Contents

The Control Program Facility (CPF) is the system support program product for the IBM System/38. CPF is designed to complement and extend the advanced capabilities of the System/38 machine to provide fully integrated support for the use of interactive, work station oriented applications. To supplement the full range of support of the interactive environment, CPF also provides comprehensive support for concurrent processing in the batch environment. CPF is designed to support a wide range of operating environments. No single environment has the exclusive use of a given set of functions. Thus any user in any operating environment has access to the functions he needs.

Many of the functions of the System/38 CPF are a direct outgrowth of the system's orientation to interactive data processing. Among these functions are:

- Data base support to make up-to-date business data available for rapid retrieval from any work station
- Work management support to schedule the processing of requests from all work station users so they are satisfied quickly and independently of other work
- Application development support that allows online development and testing of new applications concurrently with normal production activities
- System operation support that allows the system operator to perform his work through the system console or another work station using a single control language, complete with prompting support for all commands
- Message handling support that allows communication between the system, the system operator, work station users, and programs executing in the system
- Security support to protect data and other system resources from unauthorized access
- Service support that allows service personnel to diagnose problems and install new functions with minimal impact on normal work flow

An installation can be operated at a basic level (for example, with only limited interactive processing) and increase the use of controls and facilities as the needs of the installation grow.

CPF functions are used directly through the use of the control language and the data description specifications. In addition, other System/38 program products (such as high-level languages and the Interactive Data Base Utilities) also use CPF functions.

System/38 is controlled through a single, consistent control language that is supported by CPF. The control language provides the operations normally associated with controlling the operation of a system, such as:

- Controlling the operation of input/output devices attached to the system
- Submitting batch jobs for execution
- Terminating the system

In addition, many advanced functions used in data processing are provided. For example, data files and programs are created, program execution is controlled, and work station users can communicate with each other by using functions requested through the control language. However, although the control language is the interface through which the functions of CPF are controlled, it is not the only interface through which CPF or the system is used. CPF provides a specialized interface, called data description specifications, through which data in the system is described to CPF. The data is accessed and updated by high-level language programs using CPF functions.

The subsequent sections of this introduction provide an overview of CPF and a description of the control language.

CPF OVERVIEW

This overview groups CPF functions into topics according to their use. These topics are described more fully in other chapters of this publication.

Object Management

The object management facilities allow objects to be grouped and located in the system. The general term *object* is used to refer to any named item (such as a program or a file) that is stored in the system. The general term is used because all kinds of objects are located in the same manner. The object management facilities allow users to name which objects they want, without needing to specify the exact storage locations of the objects.

Certain functions of CPF, which are valid for many different types of objects, can be performed through a single set of commands. For example, functions that provide security or backup copies of objects apply to all object types.

Work Management

The work management facilities provide the framework through which the system and all the work performed on the system are controlled. These facilities provide system functions needed to support a multiprogramming environment and to manage contention between jobs for main storage and other system resources. The work management facilities allow work to be submitted by the user, presented to the machine for execution, and controlled by the system operator.

Many types of work can be performed concurrently on System/38. Often, different types of work need different operating environments to operate efficiently. For example, an interactive application that is used concurrently by a number of work station users must operate in an environment that provides rapid responses to the work station user. A batch job would not need the same type of operating environment. Through the work management facilities, specialized operating environments, called *subsystems*, control the use of resources needed for different types of work. When CPF is installed, it includes susbsystems that support interactive, batch, and spooling processing. Although the work management facilities can be used to tailor subsystems to provide specialized operating environments, the system is fully operational when it is installed. By starting, controlling, and terminating subsystems, the system operator can easily control entire operating environments through the control language.

Data Management

The data management facilities support both data base files and device files. Data base data management provides the functions required for creating data base files and performing input/output operations to them. Device data management provides similar operations for devices attached to the system, including many unique functions to support the display devices.

Generally, the data in data base files or display device files is described apart from the programs that use the files. That is, the attributes of each field (such as its length, data type, and position in a record) are described in the file rather than in the program. These data descriptions are created with the use of the *data description specifications*. A specification form (similar to RPG specification forms) provides a common format for describing the data. The form provides fixed columns for frequently specified and required information and keyword specifications for less frequently specified options.

Other device files are usually described in the traditional manner where the records and fields are described in the programs that use them. The spooling functions support the usual operations for reading files from input devices and writing files to output devices so that programs using the files are not tied directly to the external devices.

Application Development

A programmer can perform most application development activities interactively, from a work station. These activities include:

- Entering source programs into the data base
- Compiling programs concurrently with normal system operations, without interrupting the normal work flow on the system
- Testing programs in a protected environment so that production files are not inadvertently changed by a program that is being tested
- Debugging a program online, using CPF-provided functions to locate program errors
- Alternating between two interactive jobs simultaneously, such as reviewing a display of a compilation listing and reviewing the value of program variables
- Correcting the program source and recompiling the program

System Management

Through CPF functions, a system operation can control the operations of jobs and subsystems, respond to system messages, and perform other operations normally performed by a system operator. These operations can be performed from any work station and are not restricted to a single person.

The security facilities allow various levels of control over the access to objects by individual work station users. As security requirements change, the control provided by the security facilities can be modified.

CPF save/restore functions allow applications and data files to be backed up concurrently with unrelated system operation. These functions can be used to maintain backup copies of system and application objects. These copies can be used to recover from system or application malfunctions.

CONTROL LANGUAGE

The control language is the primary interface to CPF and can be used concurrently by users from different work stations. A single control language statement is called a *command*. Commands can be:

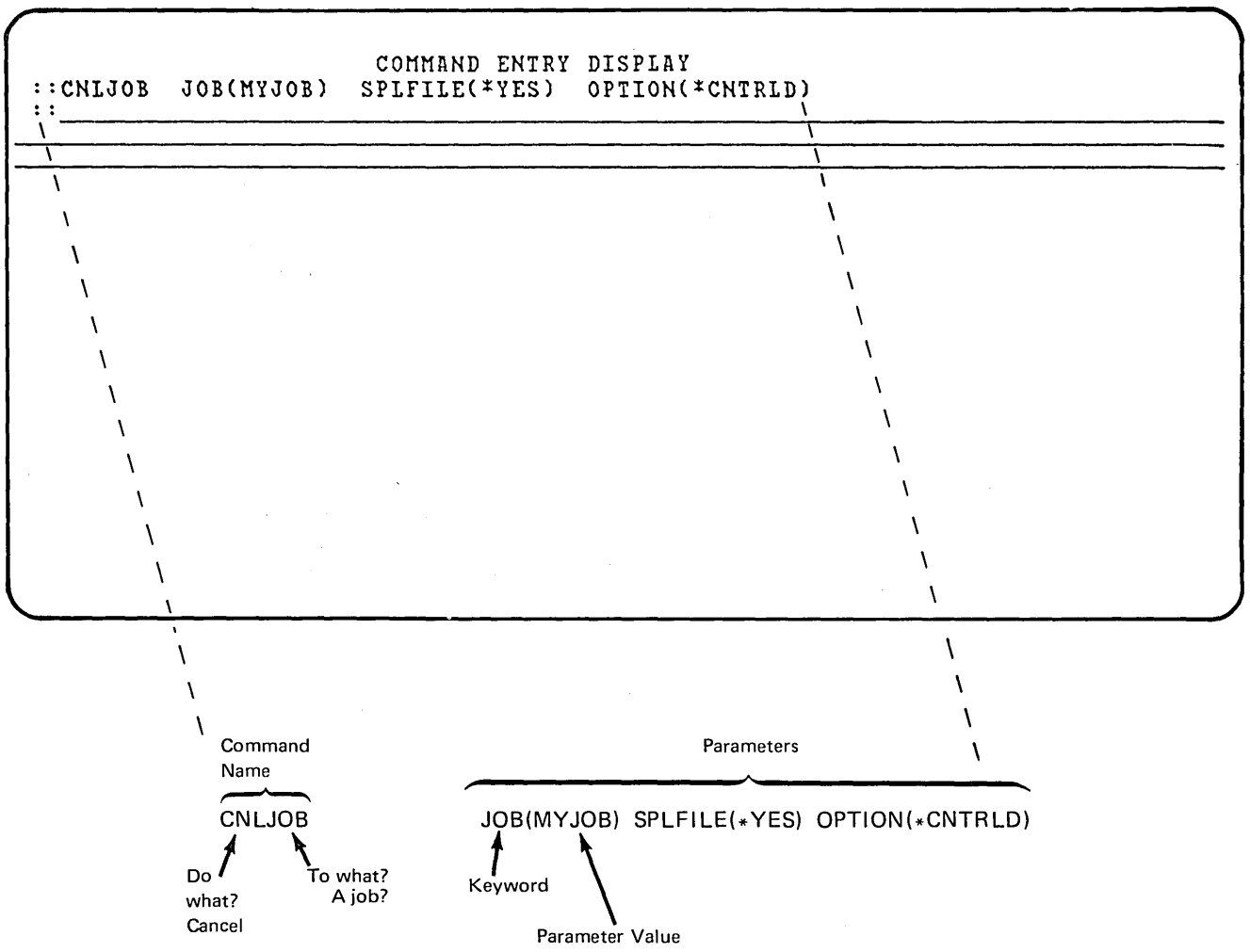
- Entered individually from a work station
- Entered as part of batch jobs
- Used as source statements to create a control language program

To simplify the use of the control language, all the commands use a consistent syntax. In addition, CPF provides prompting support for all commands, default values for most command parameters, and validity checking to ensure that a command is entered correctly before the function is performed. Thus the control language provides a single, flexible interface to many different system functions that can be used by different system users. The use of commands to create control language programs is described in *Application Development Facilities* later in this publication.

Command Syntax

Each command is made up of a command name and parameters. A command name usually consists of a verb, or action, followed by a noun or phrase that identifies the receiver of the action. The words that make up the command name are abbreviated to reduce the amount of keying that is required to enter the command. For example, one of the control language commands is the Cancel Job command. The command name is CNLJOB. The command cancels a previously submitted job.

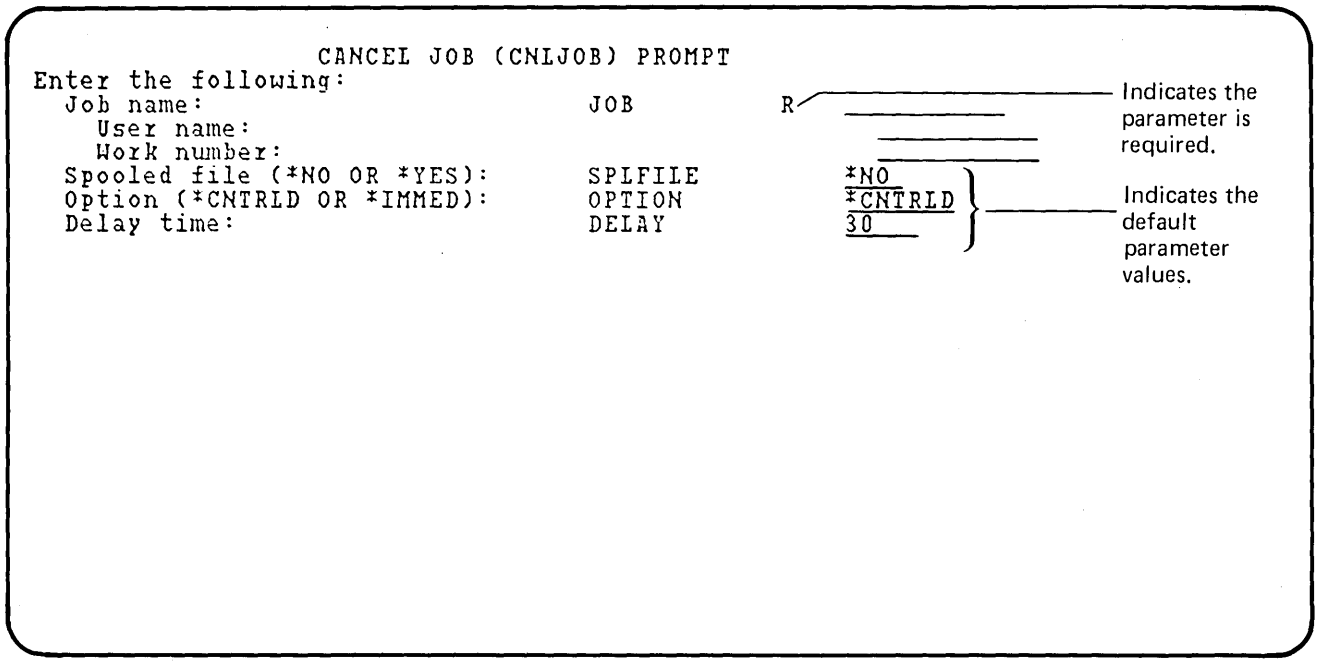
The parameters in control language commands are keyword parameters. The keyword identifies the purpose of each parameter. However, when commands are entered, the keywords are optional and can be omitted to reduce the amount of keying required. When the keywords are omitted, the parameters are positional and must be entered in the correct order. The following display shows the cancel job command entered on the command entry display and identifies the parts of a command.



Command Prompting

CPF provides interactive command prompting for any command on the system. The user can identify the command he wants to enter and then request the prompt display for the command. The resulting display consists of a set of fill-in-the-blank requests that guide the user to enter the parameter values of the command.

The following display shows the prompts for the parameters on the cancel job command.



If a command is partially entered before the prompt display is requested, any parameter values already entered are shown on the prompt display.

Parameter Defaults

Most of the parameters included in commands allow default values to be supplied by CPF if the parameter is not entered. The default value can be explicitly entered if the user desires. The prompt display provided for a command always shows any default values that are assumed if the parameters are not entered. The prompt display for the cancel job command shown earlier includes defaults for the SPLFILE, OPTION, and DELAY parameters. These defaults can be overridden by other values entered in their place.

OBJECT MANAGEMENT CONCEPTS

The object management facilities provide the functions necessary to place objects in storage and to find objects when they are needed for processing.

Objects

An *object* is a named item that is made up of a set of attributes (that describe the object) and the data portion of the object. The attributes of an object include its name, its size, the date it was created, and a text description provided by the person who created the object. The data portion of a program, for example, is the executable code that makes up the program. The data portion of a file is the collection of records that makes up the file.

The functions performed by most of the control language commands are applied to objects. Some commands can be used on any type of object and others apply only to a specific type of object.

CPF supports various unique types of objects. Some types identify objects that are common to many data processing systems, such as:

- Files
- Programs
- Commands
- Libraries
- Queues

Other object types are pertinent to the System/38 CPF, such as:

- User profiles
- Job descriptions
- Subsystem descriptions

Different object types have different operational characteristics. These differences make each object type unique. For example, because a file is an object that contains data, its operational characteristics differ from those of a program, which contains instructions.

Each object has a name. The object name and object type are used to identify an object. The object name is explicitly assigned by a user when he creates an object. The object type is determined by the command used to create the object. For example, if a program were created and given the name OEUPDT (order entry update) the program could always be referred to by that name. CPF uses the name (OEUPDT) and object type (program) to locate the object and perform operations on it.

Libraries

A *library* is an object that is used to group related objects and to find objects by name when they are used. Thus, a library is a directory to a group of objects. Libraries can be used to group the objects into any meaningful collection. For example, objects can be grouped according to security requirements, backup requirements, or processing requirements. The number of objects contained in a library and the number of libraries on the system are limited only by the amount of storage available. Thus, the number of libraries on a system can be tailored to the way the objects are used.

The object grouping performed by libraries is a logical grouping and does not affect an object's placement in storage. Thus, objects in a library are not necessarily adjacent to each other. The size of a library, or any other object, is not restricted by the amount of adjacent space available in storage. The system finds the necessary storage for objects as they are stored in the system. If an object increases in size, the system automatically allocates additional storage to the object.

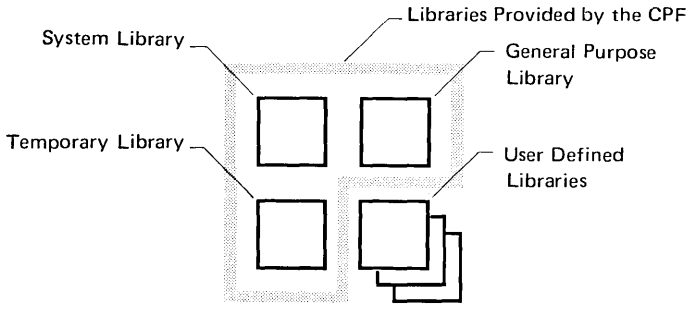
Most types of objects are placed in a library when they are created. A library cannot be contained in another library. An object can be moved from one library to another, but it cannot be in more than one library at the same time. When an object is moved to a different library, the object is not moved in storage, but it is located through the new library. Until an object is deleted from the system, it is in one library.

A library name can be used to provide another level of identification to the name of an object. As described earlier, an object is identified by its name and type. The name of the library further qualifies the object name. The combination of an object name and the library name is called the *qualified name* of the object. The qualified name tells the CPF the name of the object and the library it is in. For example, the qualified name of the program OEUPDT in the order entry library (OELIB) is OEUPDT.OELIB.

Two objects with the same name and type cannot exist in the same library, but they can exist in different libraries. Because of this, a program that refers to objects by name can be used to process different objects (residing in different libraries) in different executions of the program without any changes to the program. Also, a work station user who is creating a new object does not need to be concerned about names used for objects in other libraries. The name of the new object need be unique only in the library it is being stored in.

An object is identified within a library by the object name and type. Many of the commands in the control language apply only to a single object type, so the object type does not have to be explicitly identified. In the commands that can apply to many types of objects, the object type must be explicitly identified.

As shown in the following drawing, there are CPF-provided libraries and user-defined libraries.



The CPF-provided libraries are:

- *The system library* containing the objects that are provided as part of CPF.
- *The general-purpose library* containing user-oriented objects provided by CPF and user-created objects that are not explicitly placed in a different library when they are created.
- *A temporary library* for each job. This library is assigned to a job when the job begins. Objects created by the job can be placed in this library and are then available only to that job. The objects in this library are deleted when the job ends.

Finding Objects in Libraries

An object name can be specified as a qualified name (where both the object name and library name are specified) or as a simple object name (where the library name is not specified). If a qualified name is used, CPF finds the object in the specified library. If a simple object name is used when an object is created, CPF places the object in the general-purpose library provided by CPF. If a simple object name is used for any other operation, CPF searches a list of libraries until it finds that object name. The libraries searched and the order in which they are searched are determined by a search list called the *library list*. CPF creates an initial library list for each job when the job is initiated.

A library list has two parts. The first part specifies the libraries used for CPF functions performed during the job. This part is the same for all the jobs that run on the system and, when the system is installed, consists of only the system library. These libraries are searched before libraries in the second part of the library list. The first part of the library list cannot be changed for a job.

The second part of the library list contains the libraries that application programs use to perform their functions. When the system is installed, this part contains the general purpose library and the job's temporary library. When a system has a number of user-defined libraries, the library list might vary between different jobs. For example, for an order entry job, this part of the list might be:

1. OELIB (order entry library)

2. QGPL (general-purpose library)

3. QTEMP (temporary library for the job)

This part of the library list can be changed within a job so that the libraries used and their order can be controlled from within the job.

The use of the library list in conjunction with the use of simple object names increases the ease and flexibility of object use in System/38. When each object is created, it can be explicitly placed in the appropriate library. A library list can be designed for each job so that simple object names can be specified when the objects are used. This approach provides such advantages as:

- Easier testing of application programs. Libraries can be created to contain sample data when the program is tested. The object names used in the library are the same as those used in the normal production library. The library with the testing objects is placed before the normal production libraries in the library list. When the program has been fully tested, that library can be removed from the library list. The program then operates on the objects contained in the normal production libraries, and the object names are not changed in the program.

- Flexible use of the libraries on the system. As processing needs change, existing libraries may need to be divided into more than one library to help simplify the organization of objects on the system. This change would not require that the names of objects in the programs be changed. Only the library lists used by the jobs would need to be changed.

- The ability to let different system users operate on different objects using the same application program. Separate libraries can be created for each different user or group of users. The library list for each user's job ensures that the correct objects are used by the program for each system user.

Because of these advantages, qualified object names are not usually specified when existing objects are used. However, the qualified name can be used in situations where it is more efficient than changing the library, list or where specific objects should be specified to ensure that the correct object is used (such as in applications where security requirements are very high).

OBJECT MANAGEMENT OPERATIONS

Object management operations include general object operations and library operations.

General Object Operations

The operational characteristics of objects vary depending on the type of object involved. For example, some operations that apply to files do not apply to programs. However, there is a set of operations, known as general object operations, that apply to most object types. The general object operations are as follows:

- Display an object description. This operation displays the attributes of an object or a specified group of objects. The information can also be printed. The descriptions of a group of objects can be requested by object type, by a generic name, or by generic name and object type.

- Move an object from one library to another. This operation moves an object out of its current library and into a different library. After the operation is complete, the object can no longer be accessed through the original library. This operation is not valid for all object types.

- Rename an object. This operation changes the name of an existing object. The object itself does not change, nor does it change libraries.

- Security. These operations provide functions that control user access to an object and protect the object owner's rights.

- Save/restore. These operations provide the functions to save copies of objects offline and restore them to the system. These functions can be used to provide backup copies of objects that can be used for recovery procedures.

Library Operations

CPF also provides operations used for managing libraries on the system. These operations provide a means to observe and manage the contents of libraries as well as to control the existence of the libraries. The library operations are as follows:

- Create or delete a library. The operation for creating libraries provides the functions necessary to create user-defined libraries. After a library is created, objects can be created in it or moved into it. The delete operation is used to remove libraries from the system. When a library is deleted, any objects in it are also deleted.

- Clear a library. This operation deletes objects from a library but does not delete the library.

- Display the contents of a library. This operation displays a list of all the objects in the library or libraries specified. The information can also be printed.

- Save/restore. These operations save and restore a copy of the library and all the objects contained in it. Saved libraries can be used to provide backup copies for recovery.

The work management facilities of the System/38 CPF provide a framework through which work flow and resource usage on the system is controlled. This single framework handles all types of work performed by the system regardless of the sources from which the work is submitted. The work management facilities can be adapted to a wide range of application environments and can satisfy the requirements of diverse applications that are active concurrently.

When CPF is installed, it includes support for interactive, batch, and spooling processing and can be used without modification. A full understanding of the work management concepts is not needed to perform basic system operation. However, if support is needed for unique processing requirements, an understanding of the concepts described in this chapter is necessary.

WORK MANAGEMENT CONCEPTS

Through the use of information that is stored in various objects, the work management facilities manage the submission, initiation, execution, and termination of work on the system. The basic unit by which work is identified on the system is a *job*. A job is a single, identifiable sequence of processing actions that represents a single use of the system. Jobs are generally submitted to the system interactively from a work station (interactive jobs) or as predefined jobs (batch jobs). This second type of job is referred to as a batch job because it includes batch jobs as they are supported on other IBM systems.

An *interactive job* is a job in which the processing actions are performed by the system in response to input provided by a work station user. During the job, a dialog exists between the user and the system. An interactive job consists of all the work performed as a result of input received from the time the work station user signs on until he signs off. This work might involve processing actions performed after the user signs off, such as writing spooled output. However, the input causing these actions is received while the user is signed on.

A *batch job* is a job in which the processing actions are submitted as a predefined series of actions to be performed without a dialog between the user and the system. The job consists of all the processing actions that result from input contained within the job. Batch jobs are placed on a queue, called a *job queue*, when they are submitted to the system and are selected from the queue by the work management facilities.

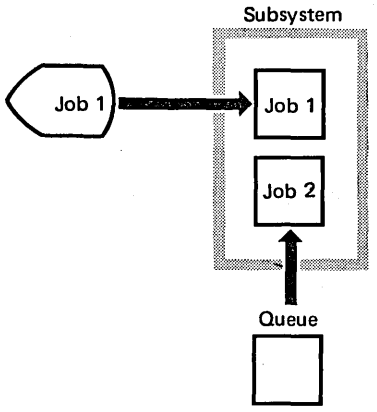
Within a job, any number of related or unrelated functions can be performed. On many systems, the execution of programs within a job is controlled through the use of job steps, which are identified in the control statements that make up the job. However, in System/38, programs can call other programs by using control language commands contained in the calling program. Thus the job is simply made up of the sequence of processing actions a system user wants performed. The functions might be requested in:

- A series of control language commands
- A single program
- One or more applications that are each made up of a series of programs

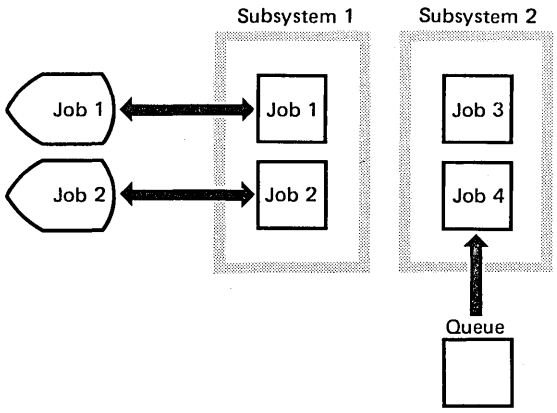
All jobs processed in the system execute within an operating environment called a *subsystem.* A subsystem is a predefined operating environment through which CPF coordinates work flow and resource usage. The specifications that define the subystem, and that CPF uses to control the subsystem, are contained in an object called a *subsystem description.*

The environment provided by a subsystem includes main storage usage, sources from which jobs can be accepted, and programs that can be invoked in the subsystem. These programs can, in turn, call other programs to perform the functions required. Although these elements are predefined, the environment provided by a subsystem does not limit the flexibility of jobs that operate within it. For example, a subsystem does not limit the files that are available nor the system functions that can be used. These are controlled by the application design and can be further controlled by the system security functions.

One or more subsystems are always active after CPF has been started. Both batch and interactive jobs can execute in one subsystem, or separate subsystems can be used for each type. The following drawing shows a system operating with one active subsystem. Job 1, an interactive job, is being used by a work station user. Job 2, a batch job, was selected from the job queue and is operating concurrently in the same subsystem.



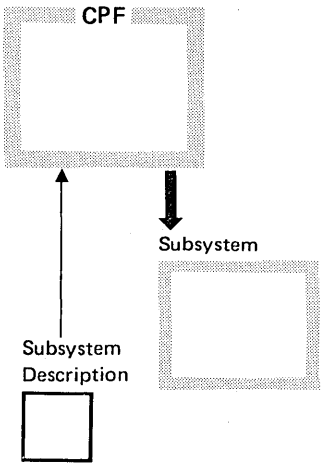
In the following drawing, two subsystems are active concurrently. Interactive jobs 1 and 2 are executing in subsystem 1. Two batch jobs, selected from the job queue, are executing in subsystem 2.



Subsystems

The use of subsystems provides the ability to establish as many, or as few, unique operating environments as necessary to meet the processing needs of an installation. The number of subsystems that can be active and process efficiently at one time is limited only by the resources available on the system. Although any number of subsystems can be active concurrently, only one active subsystem is required at any time. Each subsystem can be started and controlled independently of other work being performed on the system.

A subsystem description defines each subsystem used on the system. As shown in the following drawing, CPF uses information contained in the subsystem description to define the environment provided by the subsystem.



When CPF is installed, it includes subsystem descriptions that support interactive, batch, and spooling processing. If specialized support is needed to support unique processing requirements, the CPF-provided subsystem descriptions can be altered or user-defined subsystem descriptions can be created.

The number of subsystems an installation needs to define or to have active concurrently depends on several factors, such as:

- The number of unique processing environments that are needed to manage the use of resources by various applications
- The level of operational control that is needed over the sets of applications that operate within the different subsystems
- The amount of isolation that is needed between the sets of applications to ensure that the applications have the resources they need

Once a subsystem description has been created, the subsystem can be started and terminated by control language commands. The implementation of subsystems provides the following advantages:

- The processing environment for a set of applications can be prespecified. As processing requirements change, these specifications can be modified to meet those requirements.
- A predefined processing environment can be easily started, controlled, and terminated.
- The use of main storage and the number of jobs executing in a subsystem can be controlled while the subsystem is active so that work load changes on the system can be accommodated.
- A level of predictability can be achieved within a subsystem because the use of resources by that subsystem is isolated from the use of resources by other subsystems. This can ensure that applications with high performance requirements, such as interactive applications, have the resources needed to operate efficiently.

A subsystem description contains the following three categories of information:

- Subsystem attributes, which specify the amount of main storage available to the subsystem and the number of jobs that can execute concurrently in the subsystem
- Work entries, which specify the sources from which jobs can be accepted
- Routing entries, which specify the programs that can be invoked in the subsystem

Subsystem Attributes

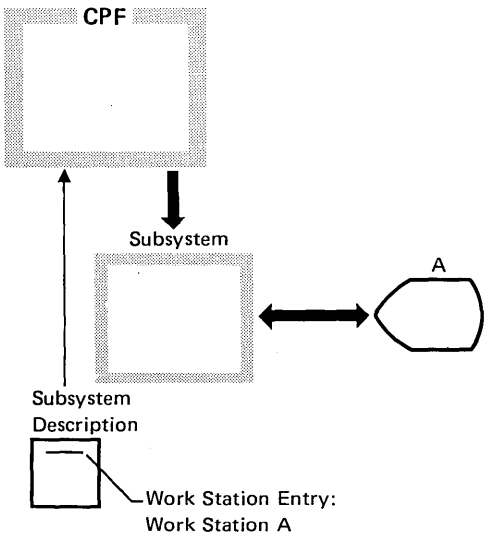
The amount of storage that the jobs in a subsystem can use is specified in the subsystem attributes through storage pools. A *storage pool* provides a quantity of main storage that can be used by jobs that execute in that storage pool. Concurrently executing jobs sharing a storage pool compete for the use of this storage but do not compete for storage with jobs executing in other storage pools. A storage pool does not acquire a block of main storage for the subsystem. Rather, it specifies an *amount* of storage. A storage pool also has an *activity level* specified for it that limits the number of jobs that can execute concurrently in the storage pool. A subsystem description can define up to 10 storage pools for a subsystem. The subsystem attributes also specify the maximum number of jobs that can be active concurrently in the entire subsystem.

Work Entries

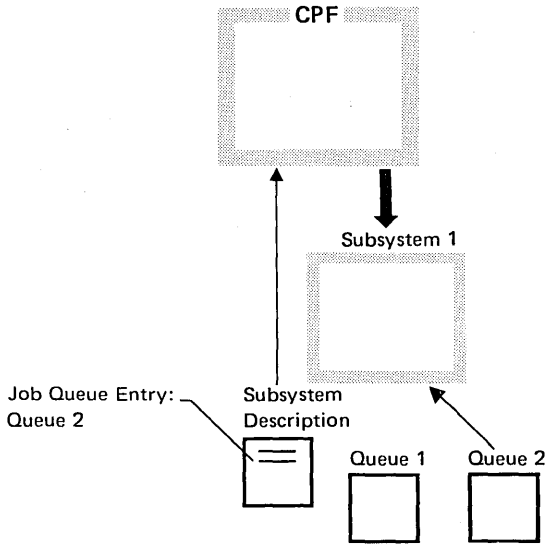
Three types of work entries can specify sources from which jobs can be selected:

- Work station entries
- Job queue entries
- Autostart job entries

Work Station Entries: Work station entries specify the work stations from which interactive jobs can be initiated. When the subsystem is started and the work station is allocated to the subsystem, the sign-on prompt is displayed. For example, in the following drawing, work station A is specified as a work entry for subsystem 1 in the subsystem description.

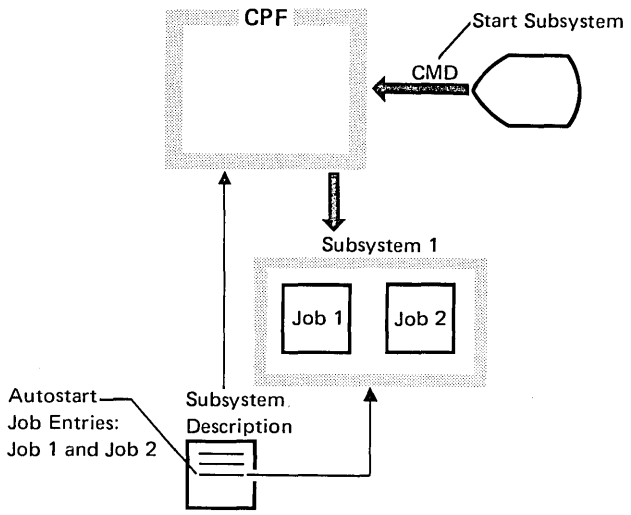


Job Queue Entry: A job queue entry specifies the job queue from which the subsystem can select batch jobs. In the following example, the job queue entry assigns job queue 2 to subsystem 1.



Jobs are placed on a job queue when they are read by a spooling reader or when they are submitted to the queue by another job. The subsystem initiates these jobs by selecting them from the job queue. Only one job queue can be specified in a subsystem description. The same job queue cannot be specified for two active subsystems. The job queue entry in the subsystem description can also specify the maximum number of jobs that can be active concurrently from the queue.

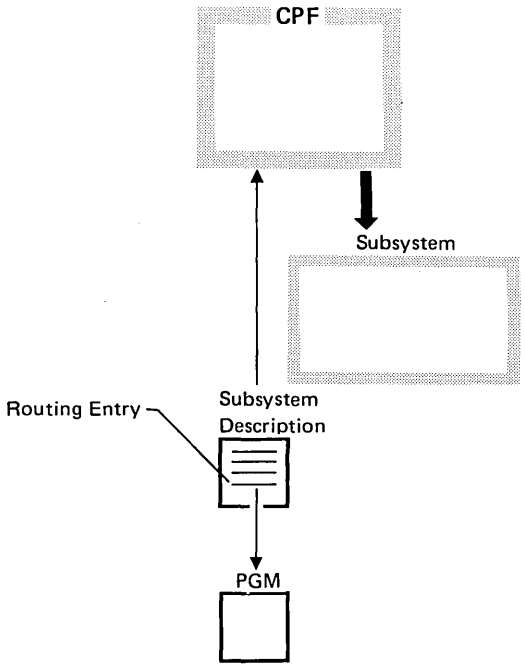
Autostart Job Entries: Autostart job entries specify any jobs that are to be automatically initiated when the subsystem is started. The jobs specified as autostart jobs do not require a work station associated with them, so they are considered to be batch jobs. In the following example, jobs 1 and 2 are started automatically when subsystem 1 is started.



Examples of autostart jobs are jobs that perform initialization functions for applications or jobs that are used as service jobs by other jobs that execute in the subsystem.

Routing Entries

A routing entry in a subsystem description specifies the program to be invoked to perform the processing for a job and the execution environment for the program.



The processing performed as a result of invoking the program is called a *routing step*. A job can consist of more than one routing step. The processing within a routing step is controlled by the program that is invoked when the routing step is initiated. That program might invoke other programs to perform the functions that make up the routing step. For example, the subsystem descriptions provided by CPF specify the CPF command processing program in the routing entries. Thus routing steps initiated in these subsystems operate under the control of that program, and work is submitted through control language commands.

Routing entries can specify user-written programs to be selected by means of *routing data*. Routing data is a character string that CPF compares with character strings in the routing entries to select the routing entry to be initiated for the routing step. Routing data can be provided by a work station user, specified in a command, or provided through the work entry for the job. The routing entries in the subsystem description might be used only once within a job (when the job is initiated). However, routing entries can also be used to change a job's environment during the job's execution.

Jobs

The user can easily manage the work load on the system by starting and terminating the subsystems needed for the various kinds of work performed. However, because jobs are separate entities that can be individually controlled, the work load can be further managed at the individual job level.

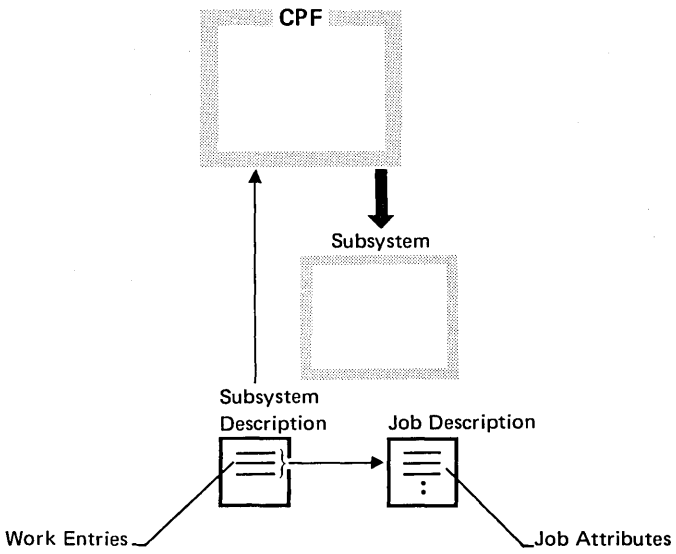
Although interactive and batch jobs appear to be quite different to the system user, once started all jobs are handled essentially the same by the system. Each job exists and is identifiable in the system from the time it is submitted (for example, from sign-on from a work station or from the time a job is placed on a job queue) until all processing actions related to the job are completed (such as writing spooled output files). As long as the job exists in the system, control language commands are available to control that job. These commands along with the commands available to control the system and subsystems provide a complete set of commands for controlling work in the system.

Job Description

Each job has a set of attributes. Different sets of attributes can be predefined for different jobs to meet the special requirements of each job. In addition, these attributes can be modified during job execution as processing needs change. Because specifying all the attributes each time a job is submitted would be tedious and time consuming, the CPF supports an object called a *job description* in which the attributes of a job can be stored. The attributes specified in a job description include:

- The job queue on which the job should be placed when it is submitted
- The scheduling priority used to control when the job is selected for execution
- The library list that is in effect when the job is started
- The routing data used by the subsystem to determine the appropriate routing entry for the job
- The default output queue onto which spooled output should be placed

Each work entry in a subsystem description except the job queue entry refers to a job description for job attributes.

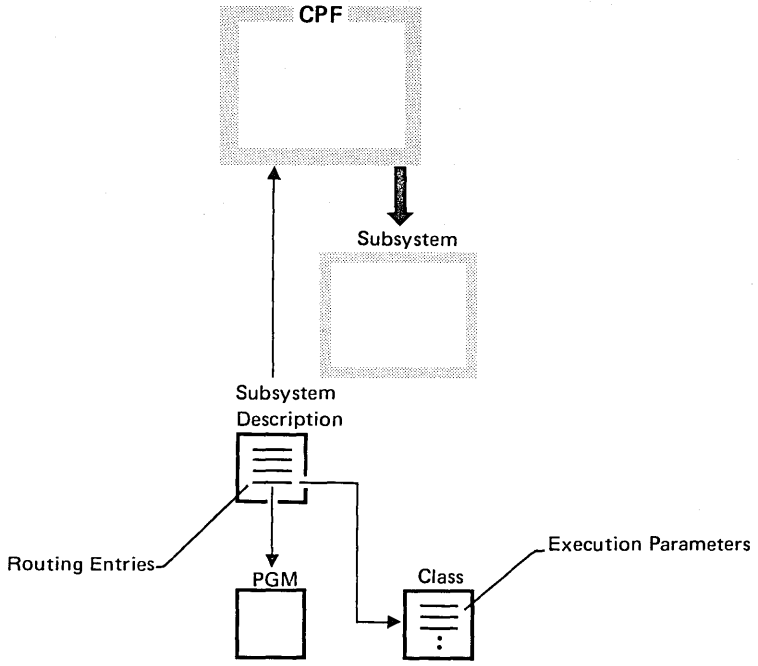


For jobs placed on a job queue, the job description is specified when a job is submitted to the queue. In this case, the job attributes specified in the job description can be overridden when a job is submitted. For example, a different library list might be specified. The library list specified in the job description would then be ignored.

Routing Steps

The job description defines the attributes (that is, the external characteristics) of a job. Certain elements of the execution environment within the machine are obtained from the routing entry. The user can change these elements while the job is executing by initiating additional routing steps in the job. A new routing step occurs whenever the subsystem uses routing data to select a new routing entry. Each routing step provides a boundary within the job at which the execution environment can be changed.

The execution environment of a routing step is defined through the routing entry that initiates the step. Some of the parameters that define this environment are contained in an object called a *class*, which is referred to by the routing entry.



The same class can be specified for any number of routing entries. The parameters that are specified in a class include:

- The machine execution priority to be given to the routing step
- The time slice, or quantity of processor time, allowed for the routing step before other waiting routing steps are given the opportunity to execute
- The maximum processor time allowed for the routing step
- A default maximum time to wait when an instruction in the routing step must wait for some resource

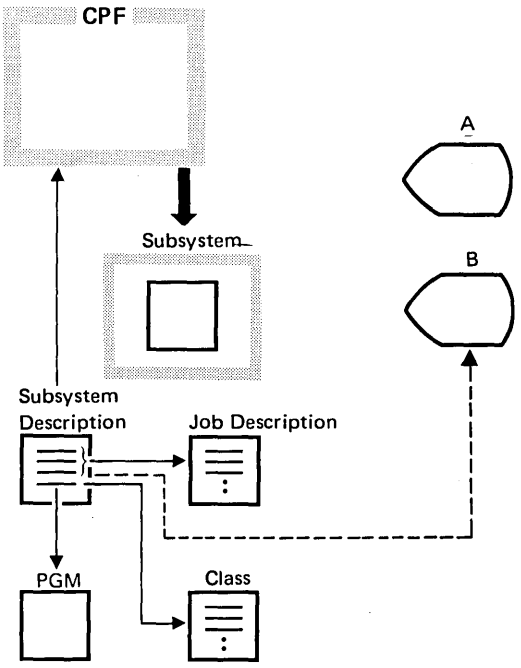
Many jobs consist of just one routing step, which is initiated when the job is started. If additional routing steps are necessary, they are requested explicitly within the job or, for interactive jobs, they occur when a work station user completes one function and requests a new function by providing new routing data to the subsystem. If more than one routing step occurs in a job, they execute serially, not concurrently.

Subsystem/Job Relationships

The framework through which the work management facilities control work is specified through definitional objects, namely subsystem descriptions, job descriptions, and classes. These objects provide extensive flexibility in managing the operational aspects of the system, namely subsystems, jobs, and routing steps.

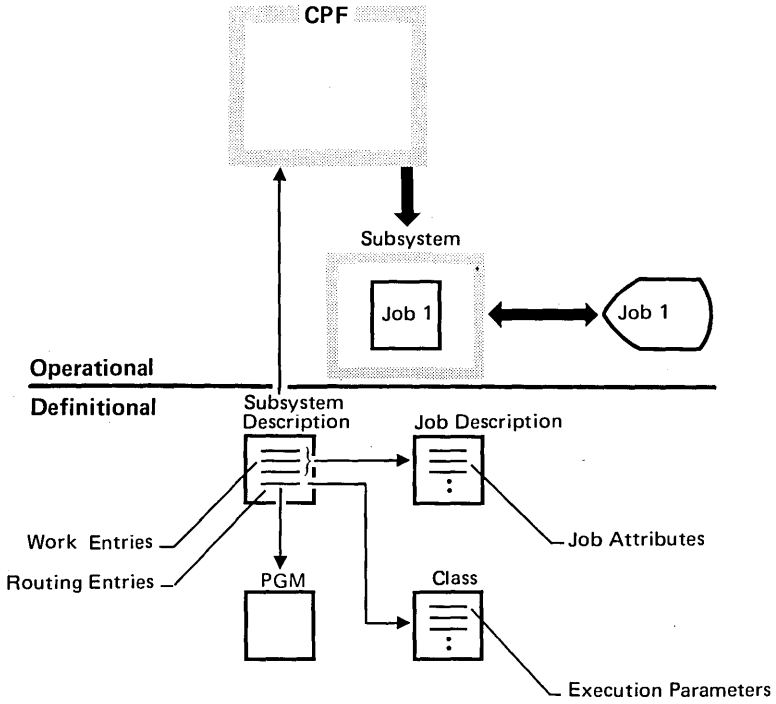
Because operating environments are defined through these objects, a single subsystem monitor is able to manage all types of environments. A separate invocation of this monitor is established for each subsystem that is started. Each invocation uses a separate subsystem description to provide the required operating environment. Because of this design, specialized programs in separate environments are not needed for special functions, such as monitoring work stations or scheduling batch jobs. In addition, most of the work management functions are driven by the occurrence of discrete events; that is, they are available when needed, but they do not use system resources when they are not being used.

The work management facilities control the execution of jobs within the subsystems that are active on the system. As shown in the following drawing, CPF uses information from the subsystem description to define the subsystem. The subsystem description refers to job descriptions, programs, and classes.



In this example, one of the work entries in the subsystem description specifies work station B as a source of work. The work entry also specifies a job description. The routing entry specifies the program and class to be used when a routing step (within a job) is started.

In the following example, a job is executing in an interactive subsystem. The definitional objects are used by CPF to provide the operational elements that exist during job execution.



WORK MANAGEMENT FUNCTIONS

CPF provides a set of objects needed for the operation of the standard subsystems, the functions needed to create or modify subsystem descriptions, and the functions needed to manage the operation of the system, subsystems, and jobs.

CPF-Provided Subsystems

When CPF is installed, it includes the subsystem descriptions that define separate subsystems for interactive, batch, and spooling jobs. These subsystems can be used as installed to satisfy normal processing requirements.

Interactive Subsystem

The interactive subsystem provides a default interactive operating environment that supports interactive jobs for all work stations. This subsystem is started automatically when the system is started and remains active as the *controlling subsystem* until the system is terminated. The interactive subsystem serves as the subsystem through which the system operator controls the system.

Because only part of the specifications in a subsystem description for an active subsystem can be changed, two copies of the subsystem description for the interactive subsystem are provided with CPF. This provides a way to modify a copy of the subsystem description so that it can subsequently be used as a modified controlling subsystem with characteristics that are different from those specified by CPF.

Batch Subsystem

The batch subsystem provides a default batch operating environment. This batch subsystem is not started automatically, but can be started by a command. All the jobs processed in the batch subsystem are obtained from the job queue. Jobs can be submitted to the job queue even though the subsystem is not active. They are available for processing when the subsystem is started.

Spooling Subsystem

The spooling subsystem provides the operating environment in which the spooling readers and writers execute. This subsystem needs to be active only when readers or writers are active. The spooling subsystem and the individual readers and writers can be controlled from jobs that execute in other subsystems.

The spooling facilities support the functions commonly provided with other systems, such as:

- Performing input and output operations apart from their related jobs
- Saving entries on job queues and output queues through system termination so those entries can be processed after the system is started again
- Manipulating and displaying entries on the queues

User-Defined Subsystems

In addition to the subsystem descriptions for the CPF-provided subsystems, CPF also provides the functions needed to create, change, and delete subsystem descriptions and to add, remove, and change specific entries in them. These functions can also be used to change the CPF-provided subsystem descriptions or to create other subsystem descriptions to support any special data processing requirements. For example, special subsystems might be needed to:

- Control an application that must be continuously available and that must provide a rapid response to its users
- Provide an operating environment that must be separately controllable
- Isolate the use of system resources to an application, such as assigning a group of work stations to a subsystem that is designed to support the application

Managing Subsystems

CPF supports the following operations for managing subsystem descriptions and their contents through control language commands:

- Creating, changing, displaying, or deleting a subsystem description. The commands that create or change a subsystem description apply only to the subsystem attributes; the work entries and routing entries are added, changed, or removed through separate commands. The commands to display or delete subsystem descriptions apply to the entire description.
- Adding, changing or removing work entries. Separate commands apply to each type: autostart job entries, work station entries, or a job queue entry.
- Adding, changing, or removing routing entries from an existing subsystem description.

Managing Jobs

Two types of operations are provided for managing jobs. Object operations apply to the job-related objects that are used by work management; execution control operations apply to the execution of jobs and routing steps.

Object Operations

CPF supports the following operations for managing job-related objects:

- Creating, displaying, and deleting job descriptions
- Creating, displaying, and deleting classes

Execution Control Operations

CPF provides commands that support the following operations to control job execution:

- Changing a job's attributes. The job being changed must exist on the system as an active job, as a job on a job queue, or as a job having output on an output queue.

- Displaying a job. The display presents information about a job. The job being displayed must exist on the system as an active job, as a job on a job queue, or as a job having output on an output queue.

- Holding a job. This operation withholds the job from further processing.

- Releasing a job. This operation makes a previously held job available for further processing.

- Canceling a job. This operation removes a job from the system. If output from the job exists on an output queue, that output can also be removed from the system.

- Sign-off. This operation terminates an interactive job.

- Submitting a job from within another active job. The job issuing the command can be either an interactive job or a batch job. The job being submitted is placed on a job queue for subsequent processing as a batch job.

CPF provides commands that support the following operations for routing steps:

- Rerouting a job. This operation causes a new routing step to be initiated for the job. The current routing step is terminated. The job continues to execute in the same subsystem.

- Transferring a job. This operation is used to transfer a job to a different subsystem. When the job is selected for execution in the new subsystem, a new routing step is initiated. The command transfers the job issuing the command.

- Allocating an object. This operation allocates an object, or a group of objects, to be used by a routing step.

- Deallocating an object. This operation deallocates an object, or a group of objects, from a routing step.

Initiating Jobs

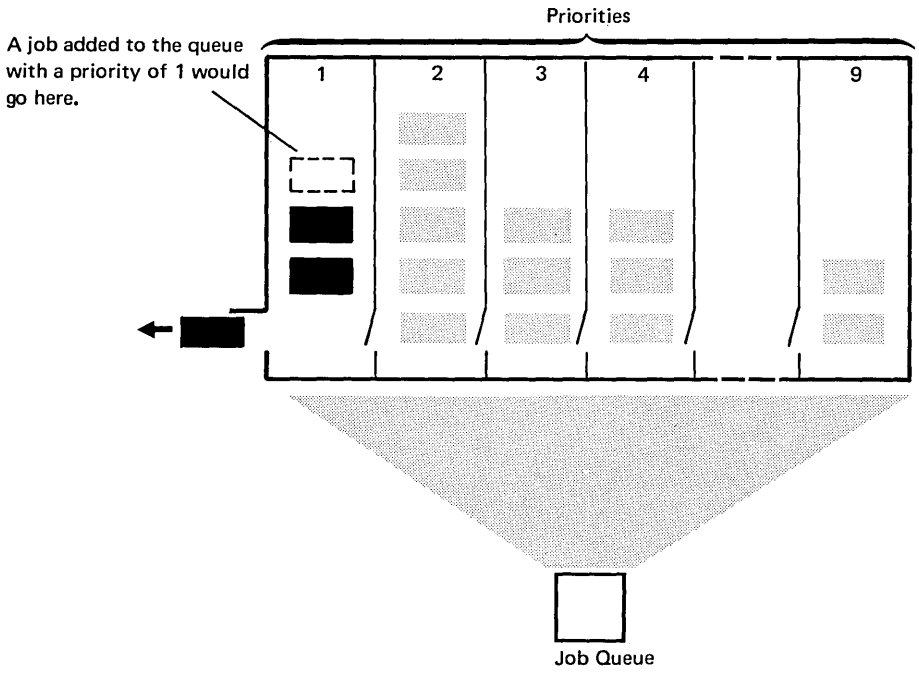
Interactive Jobs

An interactive job is initiated when a work station user signs on. An interactive job remains active until the work station user signs off or the job is terminated as a result of a command entered by another job.

Batch Jobs

Batch jobs are initiated when they are selected from a job queue. Jobs can be submitted to a job queue at any time. If the job queue is not assigned to an active subsystem when jobs are placed on it, the jobs remain on the queue until a subsystem to which the queue is assigned is started.

Jobs are selected from a job queue according to the priority assigned to each job. As shown in the following diagram, jobs with a higher priority (lower number) are selected before jobs with a lower priority (higher number). Jobs with equal priority are selected on a first-in-first-out basis.



Jobs can be submitted to a job queue from within other active jobs or through a reader.

Submitting Jobs from Within Active Jobs: Individual jobs are submitted for asynchronous execution when a need arises for a separate function that can be executed as a batch job. Jobs can be submitted to any job queue on the system. The function could be performed within the submitting job, but it might delay the work currently being done.

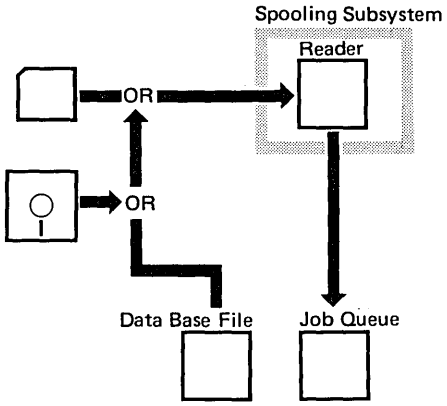
For example, if the system operator needs to save a data base file, he can simply enter the appropriate command. When the file is completely saved, he can enter other commands to continue his work. However, if the operator needs to save the file but cannot wait until the end of the operation to continue his other work, he can submit the job to be performed independently of his other work. Once he has submitted the job, he can continue with other work. The job will be selected from the job queue by the appropriate active subsystem.

Submitting Jobs Through a Reader: Some batch jobs are read by CPF programs called *readers*. These jobs can include data files, called *inline data files*, that are used by the jobs. Inline data files are placed in the system and processed as described in the *Data Management Facilities* section.

A reader reads a job from an input source and puts the job on a job queue. Each reader reads from only one source, but more than one reader can be active concurrently. The sources from which a reader can read jobs are:

- Card readers
- Diskettes
- Data base files

The following drawing shows how a reader program reads a job and places it on a job queue. The program continues reading until it reaches the end of the input or is terminated by a command.



The following drawing shows a job stream. Each job is identified by a job command. Each job might include the execution of more than one program. Programs within the job can call other programs.

```
//JOB DAILYSALES SCDPTY(2)
RPLLIBL LIBL(ORDLIB)
CALL SALES205                    /*PRINT DEPT SUMMARY*/
CALL SALES206                    /*PRINT PRODUCT SUMMARY*/
CALL SALES209                    /*PRINT SPECIAL SALES*/
//DATA
     AJ1052
     XQ4031
     BZ9504
       .
       .
       .
//JOB DAILYSHIPT SCDPTY(5)
       .
       .
       .
```

If readers are used to submit batch jobs, the efficiency of batch jobs can be increased. For example:

- Batch jobs are not limited by the speed of the device that contains the inline data files.
- The amount of device contention between jobs is reduced because each job can read input files without being constrained by the availability of the input device.
- Jobs can be read by the reader in any order because the subsystem selects jobs for execution based on priority, not on the order in which they were read.
- Named inline data files can be processed by more than one program in the same job. The file does not need to be read separately for each program that uses it.

DATA MANAGEMENT CONCEPTS

The basic elements of data management are files, records, and fields. A *file* is a collection of data records. A data *record* is a group of related data items, called *fields*. In System/38, each file has a description that describes the file, its records, and, in many cases, the fields in the records. CPF uses this description whenever a file is processed.

Files

A file is an object that is created through CPF. A file is made up of its description and the data accessed through the file. The data management facilities support two types of files: data base files and device files. All data is accessed through files.

Data base files are files whose data is stored permanently in the system. *Device files* are files whose data is read from or written to devices attached to the system. The device files supported are:

- Card files
- Diskette files
- Printer files
- Display files

The concept of a file is the same regardless of what type of file it is, which device is supported, or whether the device is attached locally or through a communication line. When a file is used by a program, it is referred to by name, which identifies both the file description and the data itself.

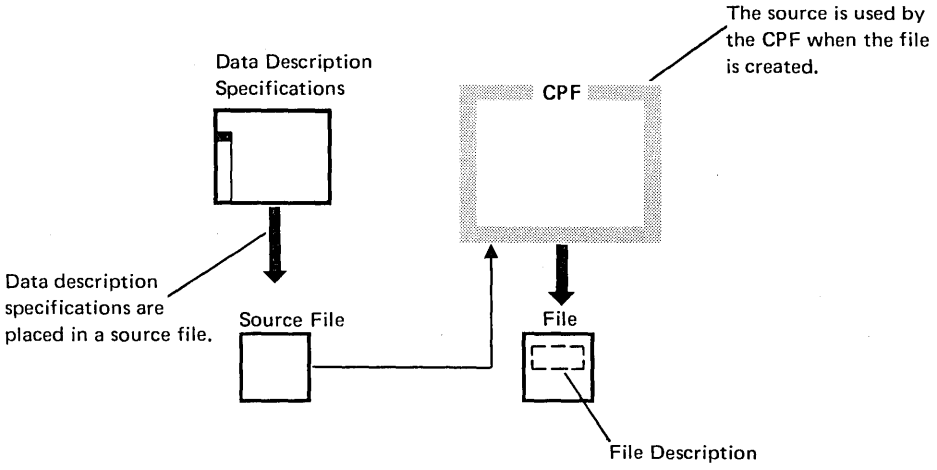
File Description

When a file is created, CPF builds the *file description* from information specified through the create command and information specified through data description specifications, if these specifications are provided. The file description contains the information necessary for a program to access a file. The file description includes:

- Data association specifications that specify where the data is in the data base or which device the file uses.
- Record format specifications that describe the format of the records contained in the file. If records of different formats are contained in the file, specifications for all the formats are included here.
- Special file attributes that further describe the file (such as whether the file contains source statements).

The information in the file description varies, depending on the type of file it describes. The kind of information contained in a data base file description is different from the kind contained in a device file description.

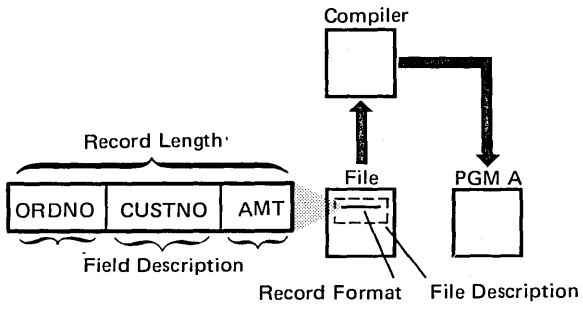
The data contained in a file is described in the file description. The record format specifications can describe the fields contained in the record. A file whose data is described at the field level in the file description is called an *externally described data* file because the data is described apart from the programs that use the file.



If the data is not described to this level, the record format specifications define the length of the record and the fields must be described in the programs that use the file. A file whose data is described this way is called a *program described data* file. Describing a file this way is similar to describing files for other systems.

Externally described data files are supported for data base files, display device files, and printer device files. Program described data files are supported for all files.

The difference between the two types exists in the location of field descriptions. In the following example of an *externally described data file,* CPF provides the record length plus the field description to the compiler when the program is compiled:



Program described data is completely described in the program.

Externally Described Data

When an externally described data file is created, the data in the file is described to CPF through data description specifications. These specifications are entered as source statements and are used by CPF to create the record format specifications associated with the file description.

When a program that uses an externally described data file is compiled, the source statements in the program specify that the record format is externally described. The compiler then uses the record format from the file description. The field names and descriptions in the record format are copied into the program by the compiler. To provide program documentation, the compiler also generates program comments from text descriptions specified in the source statements for the record format. When the file is processed, records passed between the program and CPF are made up of the fields specified in the record format.

The record format in the file description can be ignored by a program that processes the file. In this case, the record format is not copied into the program when it is compiled. Instead, the fields used by the program are defined in the program as if a program described data file were being used. The records are passed between the program and CPF according to the record format specifications, and the program divides the record into the fields defined in the program.

Programs that process externally described data files are executed in the same way as programs that process program described data files. However, there are advantages to using externally described data files, such as:

- Simplicity in writing programs that use the files. If the same file is used by many programs, the fields can be defined once to CPF and used by all the programs. This saves coding activity when programs are coded and ensures that the fields are defined consistently for all the programs that use them.

- Less program maintenance activity when the file's record format is changed. If the fields were defined in the program, the program would have to be updated whenever a change is made to the record format. When externally described data files are used, programs using the file can often be updated by changes to the file's record format without any changes to the coding in the program.

- Less redundant coding when the same record format is used by more than one file. In this case, the record format used in the first file can be referenced when subsequent files are created. This saves coding time while ensuring consistency in the record formats. This technique is especially useful when data base files use the same record format.

- Improved documentation because:
  - Programs that use the same files use consistent record and field names
  - Text and column headings can be included in the record format and displayed at the work station
  - Text descriptions of the files and records can be displayed, which supports inquiries about the files and record formats
  - Improved integrity because record format level checking avoids the use of a file whose record format does not match the record format used in the program

Program Described Data

When a program described data file is created, it contains a default record format specification. This record format defines a record of one field of character data. The field length equals the record length. When a program processes the file, records are passed to and from the program as single, character fields. The program must define any individual fields used in the record.

Special File Attributes

The special file attributes in a file description specify such things as:

- Whether record format level checking should be performed when the file is processed
- Whether the file is a source file
- Whether a device file is to be spooled
- Whether the file can be shared by more than one program while the file is being processed

Record Format Level Checking: Record format level checking provides a way to ensure that the record format specifications have not changed between the time the program was created and the time it is executed. Record format level checking can be specified when the file is created or when a program that uses the file is executed.

To support record format level checking, CPF assigns a record format level identifier to each format in an externally described data file. The level identifiers for each record format are placed in the program when it is created. When the program processes the file, the level identifiers in the program are compared with the level identifiers in the file to ensure that the record format has not been changed since the program was created.

Source File: A source file is a file created to contain source statements for such items as:

- High level language programs
- Data description specifications
- Command definitions
- Print images
- Translation tables

Source files use the same record format, regardless of whether they are data base files, card files, or diskette files. The record format for a source file specifies a sequence number field and a date field followed by the data (source statement).

Spooled Files: A device file defined as a spooled file is not intended for direct access to a device. Instead, these files provide access to data processed by the readers and writers. When spooled device files are opened for input, they provide access to data read inline with a job. When they are opened for output, the data is stored by CPF until it is written to the device by a *writer*.

Spooling can be specified for card, printer, and diskette device files.

Connecting a File to a Program

A file is connected to a program when the file is opened. Opening a file requires that the file be allocated either explicitly as a result of a control language command requesting the allocation or automatically as part of the open operation. Data base files can be allocated exclusively to a program or shared by different jobs. Explicitly allocating a file to a job (before the file is opened) ensures that the programs in the job will have access to the data when the file is opened.

When a program opens a file, CPF creates a data path between the program and the file. This data path allows data to be passed between the file and the program. For data base files and spooled device files, the data path connects the program to data stored in the system. For nonspooled device files, the data path connects the program to the device associated with the file. The data path is maintained until the file is closed by the program or until the routing step ends and the file is automatically closed by CPF.

File Overrides

File overrides are parameters specified when a file is used to temporarily change parameters specified when the file is created. Any overrides specified for the file are performed before the file is opened. Overrides are specified through control language commands and can be used to change such items as:

- The name of the file to be processed
- Whether output is to be spooled
- Whether record format level checking should be performed

File Processing

When a program is connected to a file, the program can perform input/output operations to use the file. The program can be written so that it is independent of the type of file being processed or is dependent on the file type.

File independent programs are written so that the input/output operations performed are not unique to the type of file being processed. For example, a program that sequentially reads all the records from an input file is not dependent on the file type. In one use of the program, it might read records from a card file. In another use, as a result of an override, the program might read records from a data base file. CPF performs the operations requested for each type of file. File independent programs can process different files from the same type of device or from different device types at different times.

The use of file independent programs provides additional flexibility in the way files and devices are used on the system. For example:

- A program that normally processes a card input file can be used to process a diskette or data base file.
- A program that normally produces output to a printer file can produce output to a diskette file.
- A program that normally produces output to the system printer can produce output to a work station printer.

In any of these cases, the program would execute normally regardless of the file type involved in a particular program execution.

On the other hand, file dependent programs are written to take advantage of the full range of operations that are valid for the type of file specified in the program. These operations might not be valid for other file types. For example, various retrieval methods are supported for data base files. Also, for display device files, many unique functions are supported for device operations, such as display formatting and the use of command function keys.

DATA BASE DATA MANAGEMENT

The System/38 data base includes all of the files whose data is stored permanently in the system. Data base data management lets different programs operate on data independently of other programs using the same data. Each program can view the data in a way that meets the requirements of that program. Additional data can be added to the data base without affecting the use or availability of the data that is already there.

Data base files provide permanent data storage much as normal disk files do on other systems. However, because of the data base facilities and because the data is permanently online and can be used concurrently by many applications, the following advantages are achieved:

- The applications have constant access to up-to-date data because data that is updated is immediately available to other applications that use it.
- The concurrent availability of data to more than one program is improved because the data can be locked by record when a program uses it, and thus the entire file does not have to be locked to the program.
- Storage space is used more efficiently because less data redundancy exists on the system.
- System operation is more efficient because separate jobs do not have to be run to sort and update data files every time an application is going to use them.

Basic to System/38 data base data management are the concepts of access paths, physical files, and logical files. Access paths provide the organization necessary to process the data stored in data base files. Two kinds of files are used to process the data. One actually contains data and is called a physical file. The other, called a logical file, provides alternative methods of formatting and accessing data that is stored in one or more physical files. Application programs are written the same way regardless of whether they use physical or logical files. In either case, the program simply processes a file.

Access Paths

An *access path* provides a logical sequence to records that are stored in data base files. In many previous systems, records were stored on the disk according to the organization dictated by the access method. In System/38, records are stored independently of their retrieval organization. When records are added to a data base file, they are stored in a physical file in the order of their arrival. When the records are processed, CPF uses the access path as the means by which to locate and retrieve the data records needed by a program, either randomly or in a predefined sequence. The access path to be used is specified when the file is created. When the file is used for input/output operations, records are processed according to the sequence provided by the access path. When the file is processed, the access path can also be used to select or omit certain records.

More than one access path can be used to access the same physical data in the data base. The access paths can be maintained as the files are processed so that when the data is used concurrently by more than one program, each access path to the data reflects the current contents of the file. This support lets different users randomly access and update the same data through different keys. Alternatively, access paths can be rebuilt when they are opened; this method avoids the overhead of dynamic maintenance.

CPF supports two types of access paths: arrival sequence and keyed sequence.

Arrival Sequence Access Path

This access path is based on the order in which the records are stored in a physical file. Records in the file can be processed:

- Sequentially, where the records are retrieved consecutively from the file
- Directly by relative record number, where the record is identified by its position from the beginning of the file or from the last record accessed in the file

Processing files using the arrival sequence access path is similar to processing consecutive or direct files on previous systems.

Keyed Sequence Access Path

This access path logically organizes the records in a file according to the contents of key fields in the records. A *key field* is a field whose contents are used to sequence the records in the file. Records in the file can be organized according to either the ascending or descending sequence of the key fields. Records in the file can be processed:

- Sequentially, according to the contents of the key fields
- Randomly, with the key fields identifying the records

A keyed sequence access path is created when the file is created. It is updated whenever records are added to or deleted from a file or whenever a record is modified and the contents of a key field change. Thus all the access paths to the data can be maintained concurrently.

Key Fields: The contents of more than one field can be used as a single key. In this case, all the records that have the same value in the first key field are sequenced by the contents of the next key field, and so on. The fields used in the key can occur anywhere in the record format. In addition, some of the fields might be used in ascending sequence and others in descending sequence. Duplicate key fields can be allowed, and specifications for processing them can be included when the file is created.

Processing Keyed Sequence Files: Using a keyed sequence access path is similar to using an indexed sequential access method on previous systems. Records in the file can be accessed sequentially or randomly. Random processing includes the capability to select a record randomly, then to process succeeding records sequentially. Thus, a work station user (through an application program) could randomly select a record, then continue with sequential processing.

If more than one key field is used for the keyed sequence access path, either the entire key or a partial, or *generic*, key can be used to retrieve the records. If a generic key, which has fewer fields than the entire key, is used to retrieve the record, the first record that contains the requested key field is retrieved from the file.

Members

Data records within a data base file are grouped into *members*. All the records in a file can be in one member or they can be grouped in different members. The first member can be named when the file is created. Subsequently added members are named when added to the file. Each member of a file is processed individually through a separate access path. The records from different file members are not merged. A logical file member can be used to merge records from more than one physical file member so they appear to exist in one file.

Using more than one member of a file to contain different groups of records requires less system overhead than creating a separate file for each group. Examples of files with more than one member include:

- Source files in which each member contains the source statements for a different program
- An order file that has a separate member for each month's incoming orders

Physical Files

A *physical file* is a data base file that contains data records. Thus a physical file is similar to files on disk for other systems. All the data records in a physical file have the same format. That is, a physical file contains fixed-length records, all of which contain the same fields. CPF stores records in a physical file in the order in which they are placed in the file. However, the records can be processed in any order that is established by the access path specified when the file is created.

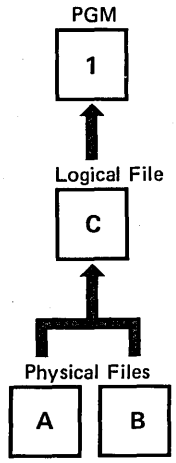
The file description for a physical file is created when the file is created. The file description includes:

- A description of the record format used by the file
- A description of the access path used for processing records from the file
- A description of the storage attributes of the file

The data contained in a physical file can be processed through many different logical files. Consequently, the records in a physical file can contain more fields than any single program would process at one time. However, because a physical file can contain the fields used by many logical files, fields that are used in more than one logical file need to be stored only once in the data base. When such a field is updated for one logical file, it is also updated for all the other logical files that use that field. Thus the storage space in the system is used efficiently, and up-to-date data is available to any programs using the data through various logical files.

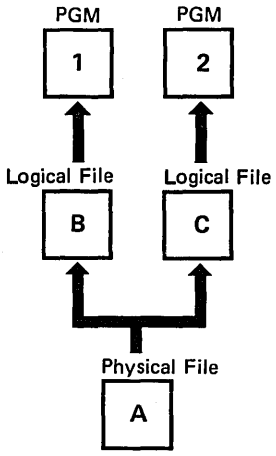
Logical Files

A *logical file* is a data base file through which data that is stored in one or more physical files can be accessed by means of record formats and/or access paths that are different from the physical representation of the data in the data base. A program that processes data by using a logical file operates as though the file actually contained the data. In the following example, program 1 is accessing data from physical files A and B through the logical file C:



A logical file can be used to access data from physical files, but not from other logical files because logical files do not actually contain data. Logical files can share the same access path.

One of the many advantages of using the data base is that different programs can access the same elements of data as they need it, through different logical files. In the following example, programs 1 and 2 both access data from the physical file A, but through different logical files (B and C).



Data in a physical file can be used by any number of logical files. Each file can impose its own characteristics, such as field length and type, on the data. However, the data is always stored in the system according to the characteristics and organization specified by the physical file. When a data record is accessed through a logical file, the data is transformed to meet the requirements of the logical file. This data transformation is not apparent to the program using the file.

Logical files can share an access path so that two or more logical files can use the same access path to retrieve the same data. For example, two logical files might use different record formats to process the same data from a physical file. If they both want the records sequenced on the same key field in the same order, they can share the same access path. When this is done, the system needs to build and maintain only one access path for the two files.

A logical file can impose its own characteristics and organization on data independently of the data's physical characteristics and organization. For example, an order entry application might put incoming orders in two physical files. One, named HDRIN, contains header information from each order received. The second physical file, named DTLIN, contains the detailed information from each order. One record is generated for each line on the order. The record formats and the records contained in these physical files are shown in the following table.

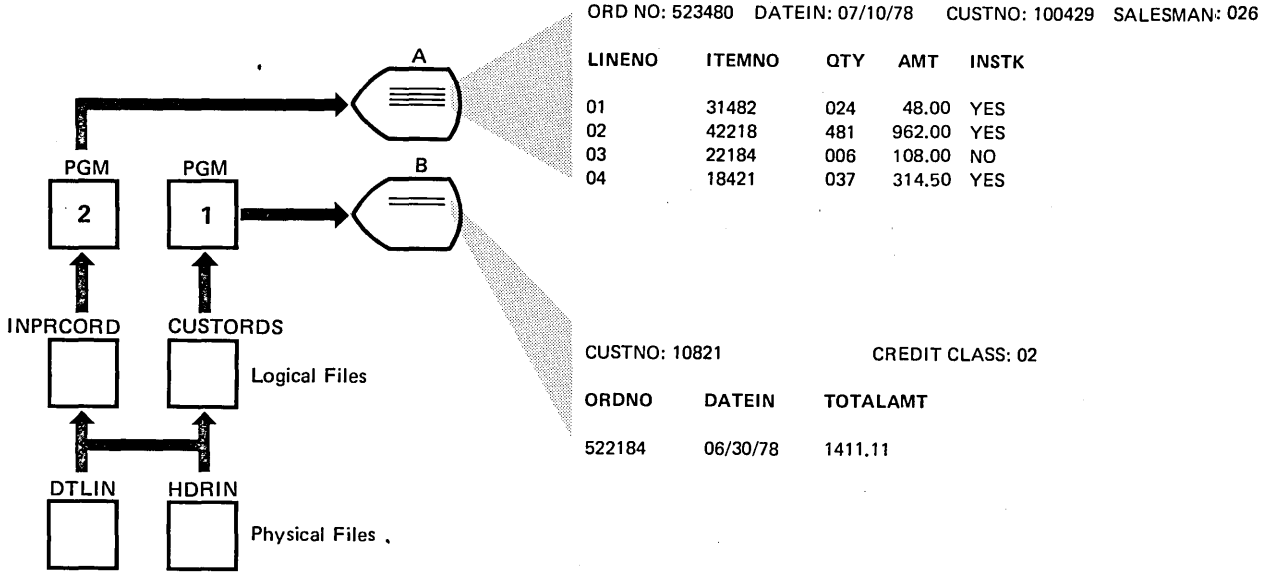
PHYSICAL FILE: HDRIN

RECORD FORMAT:	ORDNO	ORDDAT	CUSTNO	SALMN	TTLAMT
	522184	063078	108211	085	141111
	523480	071078	100429	026	143250
	534800	071078	180241	047	040479
	553672	071178	123876	079	108449
	564211	071578	156827	068	046848

PHYSICAL FILE: DTLIN

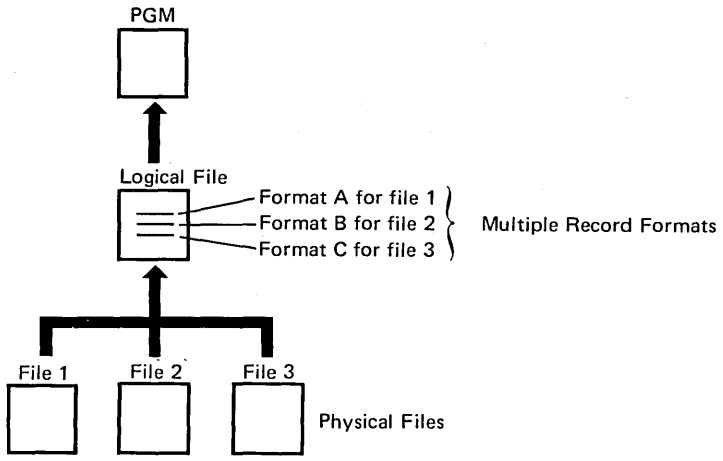
RECORD FORMAT:	ORDNO	LINENO	ITEMNO	QTY	AMT
	522184	01	46628	048	059210
	522184	02	06189	020	003660
	522184	03	17281	129	078241
	523480	01	31482	024	004800
	523480	02	42218	481	096200
	523480	03	22184	006	010800
	523480	04	18421	037	031450
	534800	01	28442	080	024548
	534800	02	41298	041	015949
	553672	01	46821	420	108449
	564281	01	80149	010	040000
	564281	32	42821	016	006848

These two physical files are shared by two logical files that are accessed by work station users. The work station users access the data through order entry application programs. In the following drawing, user A has asked to see order number 523480. The order is available through the logical file INPRCORD (in-process orders). User B has asked to see all the orders that are entered for customer number 108211. This information is available through the logical file CUSTORDS (customer orders).



Multiple Record Formats

A logical file can use more than one record format. Each logical file record format must be related to one physical file record format and can be used for retrieving data from one or more physical files. One logical file record cannot contain data from more than one physical file. Through the use of a single logical file with multiple record formats, data from more than one physical file can be processed as though the data were all in the same file, as follows:



The records processed from a logical file can vary in length because of the different record formats used. The access path for the logical file determines the order in which the records are processed from the physical files. This makes different record formats from different physical files available through one logical file.

The File Description

The file description is created when the file is created and includes:

- A description of each of the record formats used by the file
- A description of the access path used for processing records from the file
- An identification of the physical files containing the data that can be processed through the logical file

The file description and an access path are all that exist in storage for a logical file, because the data is actually contained in one or more physical files. If the logical file record format is different from the physical file record format, records processed through a logical file are transformed to the logical format by CPF as the records are retrieved by a program using the logical file.

Using Data Base Files

The System/38 data base allows information to be stored on the system in an efficient manner and also provides great flexibility in how that data is used. Through the use of physical files, logical files, different record formats, and different access paths, application programs can be designed to use whatever data they need. The data base can be organized so that a minimum of redundant data is maintained on the system.

The following example illustrates how the data base data management facilities are used. An application program processes customer orders. Two types of information are needed:

- Header information that applies to each order. This information includes the order number, the customer name, the customer address, the order date, and other information that applies to the whole order.
- Detail information that applies to each item included in the order. This information includes the item being ordered, the quantity, and the price.

In this example, each item ordered should be treated separately, as one record. The header information should not be repeated in each record, so two physical files are used. One physical file contains records of header information, with one record for each order received. The other physical file contains records of detailed information, with one record for each item included in the order. The only information that must be repeated in both files is the key fields (for example, the order number field).

To process orders, this program needs both the header records and the detail records. The program could be written to process both physical files. Instead, a logical file is used that shares the data contained in the two physical files. The logical file has two record formats, one for each type of record. The access path uses key fields in the records so that the detailed records are presented to the program following the appropriate header record.

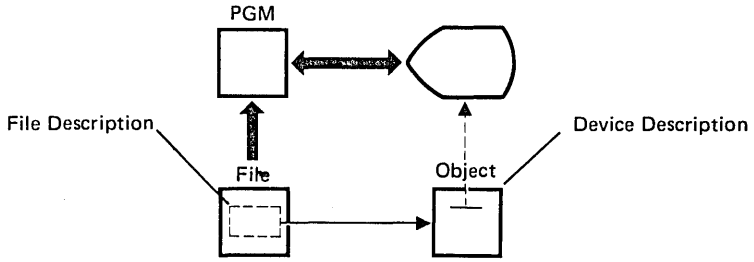
When a work station user executes the application program, he accesses the data contained in both physical files. The program lets the work station user request information about any order contained in the files. Another application program, using different logical files with a different access path, lets the work station user request information about orders placed by individual customers. In addition, programs can use either physical file independently.

DEVICE SUPPORT DATA MANAGEMENT

The device support data management facilities support the external devices that can be attached to System/38. The devices supported are the display devices (which include the console), the diskette magazine drive, the multifunction card unit, and the system and work station printers. The device file descriptions are kept in the system and are used by CPF to transfer data to and from the devices.

There is a *device description* object for each device that is attached to the system. Device descriptions for some devices are provided by CPF. Others are created when devices are defined to CPF.

CPF creates a *device file,* which includes the file description, for each device defined to the system when the device is installed and defined to CPF. The file description for any file refers to specific device characteristics in the device description. Thus, as shown in the following drawing, the file description and the device description serve as connecting links between the program and the data in the file.



Device files created by CPF are program described data files. That is, the formats of the records in these files must be described in the programs that use them. Additional device files can be created to meet any special processing requirements.

Both externally described data files and program described data files can be created for work stations and printers. However, printer files usually use program described data. Only program described data files can be created for card and diskette devices.

Display Device Support

The display device support is designed to simplify the use of display devices (work stations) by application programs and provide functions that are not easily accomplished on many interactive systems. Most display files are externally described data files. Externally described data files offer the following advantages:

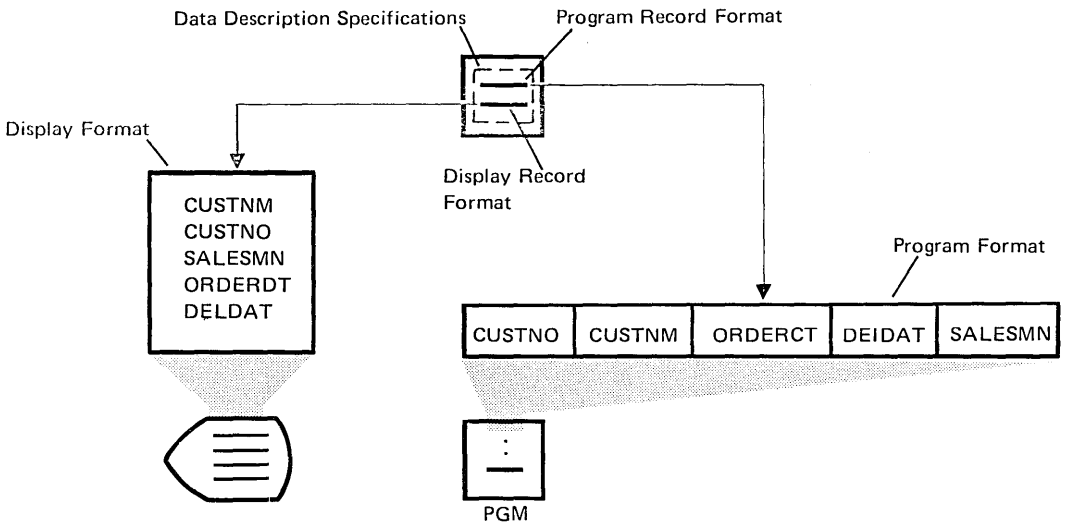
- CPF performs the device control operations, including formatting data on the screen, accepting input from the keyboard, and handling error conditions that occur at the device.

- CPF can perform subfile operations, which let the program perform input/output operations that send and receive multiple records in one operation. The program processes one record at a time, but CPF and the work station send and receive blocks of records. If more records are transmitted than can be displayed on the screen at one time, the work station operator can page through the block of records without returning control to the program.

- CPF can validate information entered by the work station user and let the user correct any errors before the record is passed to the program. For example, if the work station user enters alphabetic characters into a numeric field that is validated by CPF, the work station user would be informed of the error without returning control to the program.

- CPF can accept indicators from the program to control the operations performed at the work station and return indicators to the program to inform it of actions taken by the work station user.

When externally described data files are used for display devices, coding the application program is simplified because the program sends and receives records as they are described in the file's record formats. The record format describes both the format of the record used in the application program and the format of the record when it is displayed. The formats are described to CPF through data description specifications, as follows.



If program described data files are used for display devices, the record formats and display formatting must be specified in the application program that processes the file. The following discussion about display device support is limited to the use of externally described data files.

File Description

The file description is created when the file is created. It consists of the record formats for the file and the functions that CPF is to perform when input/output operations are requested.

When the file is processed, CPF transforms output data from the program to the format to be displayed and displays it on the screen. When data is passed to the program, CPF transforms data from the device to the format used by the program. CPF performs all the operations needed to control the work station. It also passes indicators between the work station and the program so that the work station user and the program can communicate with each other.

Record Formats

Record formats tell CPF what fields are contained in a record, how the fields should appear at the device, and how the fields should appear to the program. Thus, the record format is the basic unit for passing information between an application program and the work station user.

A record format used for a display file can contain fields used for input only, for output only, and for both output and input (called output/input fields). Output fields contain information that is displayed to the work station user. Input fields allow information to be entered by the work station user. Output/input fields contain output that can be overlaid and returned as input by the work station user.

Attributes can be defined to control the way the fields are displayed or processed. For example, field attributes allow:

- Displaying fields in reverse image
- Blinking the field
- Validity-checking input keyed into the field

Indicator fields can be specified to provide communication between the program and CPF. These indicators, which can be set on or off, can be used to control data management functions for output operations and to indicate the results of input operations. For example, an indicator used in an output operation (called a conditioning indicator) could cause a field to be highlighted when it is displayed. For input operations, an indicator called a response indicator could be used to inform the program that a specific key was pressed by the work station user.

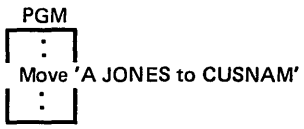
Using Display Device Support

All the operations necessary to establish an interface through which system users can communicate with application programs can be performed by the use of the display device support functions. These functions can:

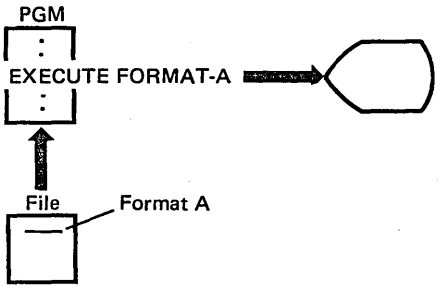
- Format displays on the screen so that the work station user can use the information provided by the program. This function includes the capability to control, from the application program, which fields are displayed on the screen.
- Design displays into which the work station user can easily enter input.
- Use subfiles for either output or input so that the work station user can work on a block of records. This capability can reduce the amount of CPF activity required between the work station user and the program. It also lets the work station user scan the records without returning to the program.
- Handle errors that occur at the work station without returning to the application program.
- Display error messages to the work station user based on indicators passed to CPF by the application program.

The following steps illustrate how an application program might process records using a display device file. The example uses functions provided by high-level language programs and data description specifications. The example detects an error in the input and uses CPF functions to display an error message.

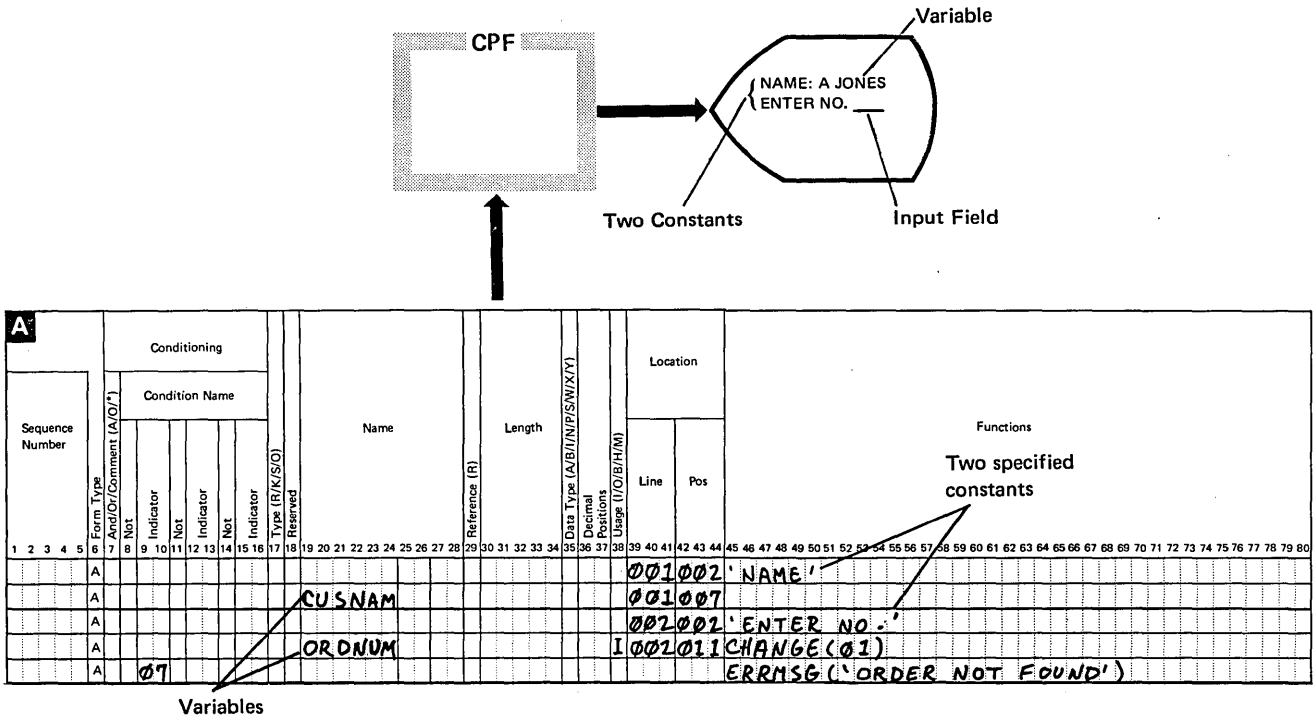
1. The application program moves the data A. JONES into a variable named CUSNAM.



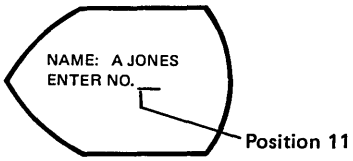
2. A high-level language application program passes an output record to the work station, using record format A.



3. Using record format A, CPF displays the record on the screen. The two constant fields (NAME: and ENTER NO.) are specified in the record format. The variable from the application program is displayed following NAME:. The input field following ENTER NO. is to be filled in by the work station user.



4. The cursor is placed at the first position of the input field (position 11), as specified in the record format.

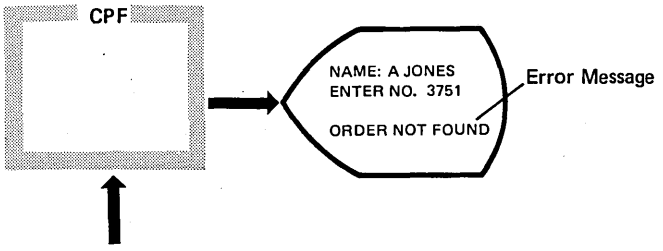


5. The work station user keys 3571 into the input field and presses the Enter key to indicate that he has finished entering the data.

6. CPF places the data, 3571, into the variable ORDNUM and turns on indicator 01 to indicate that the field has been changed.

7. If the data entered is incorrect, the program turns on indicator 07 to display an error message. The program then passes control to the CPF to display the error message caused by indicator 07.

   CPF displays the error message on the bottom line and displays the input field (3571) in reverse image with the cursor at the beginning of the input field.



A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not	Indicator	Not	Indicator	Not	Indicator	Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (A/B/I/N/P/S/N/X/Y)	Decimal Positions	Usage (I/O/B/H/M)	Location: Line	Pos	Functions
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A																001	002	'NAME'
	A										CUSNAM						001	007	
	A																002	002	'ENTER NO.'
	A										ORDNUM					I	002	011	CHANGE(01)
	A		07																ERRMSG('ORDER NOT FOUND')

Indicator

Error Message

Nondisplay Device Support

The nondisplay device support functions are the data management functions that apply to printers, the diskette drive, and the MFCU. The file description for a nondisplay device file includes:

- Identification of the device associated with the file
- Spooling and output scheduling information for the file
- Device-dependent information such as forms or card types to be used, hopper and stacker use, and the number of copies to be produced for output files

Using program described data files in your programs is similar to using files on other IBM systems. The records are described in the using programs, and record types must be identified in the program if more than one type of record can be contained in the same file.

Printer File Support

The system and work station printers attached to System/38 are supported by CPF through printer files. Among the functions supported for printed files are:

- Folding or truncating the output records if the records passed to the printer are longer than the maximum length allowed for the device. Folded records are continued on subsequent print lines until the entire record is printed.
- Allowing the operator to align printer forms before the file is printed.
- Initializing constant fields for a file.
- Editing fields in the records according to a predefined edit code or edit word.

Printer files can also be externally described data files. The record format is then contained in the file description. Using externally described data printer files increases a program's file and device independence because the file description can control such specifications as character substitution, forms alignment, and forms size. These specifications can be overridden when a program is executed if the user specifies a different externally described data printer file. In addition, using a different file can change the format of the printed records, even though the program is not changed.

Multifunction Card Unit Support

The multifunction card unit is supported for card input and output files. Among the functions supported for card files are the following:

- Output files can be punched only, printed only, or both punched and printed
- Combined files can be used that combine reading input data and punching and/or printing output data
- Output records that exceed the length allowed by the device are truncated to the maximum length allowed

Card device files are always processed as program described data files.

Diskette Magazine Drive Support

The diskette magazine drive is supported through high-level language programs for both input and output files. CPF includes the functions needed to initialize diskettes, display the volume and file label information from a diskette, rename diskettes, clear diskettes, and duplicate diskettes. Multivolume diskette files are also supported. Diskette device files are always processed as program described data files.

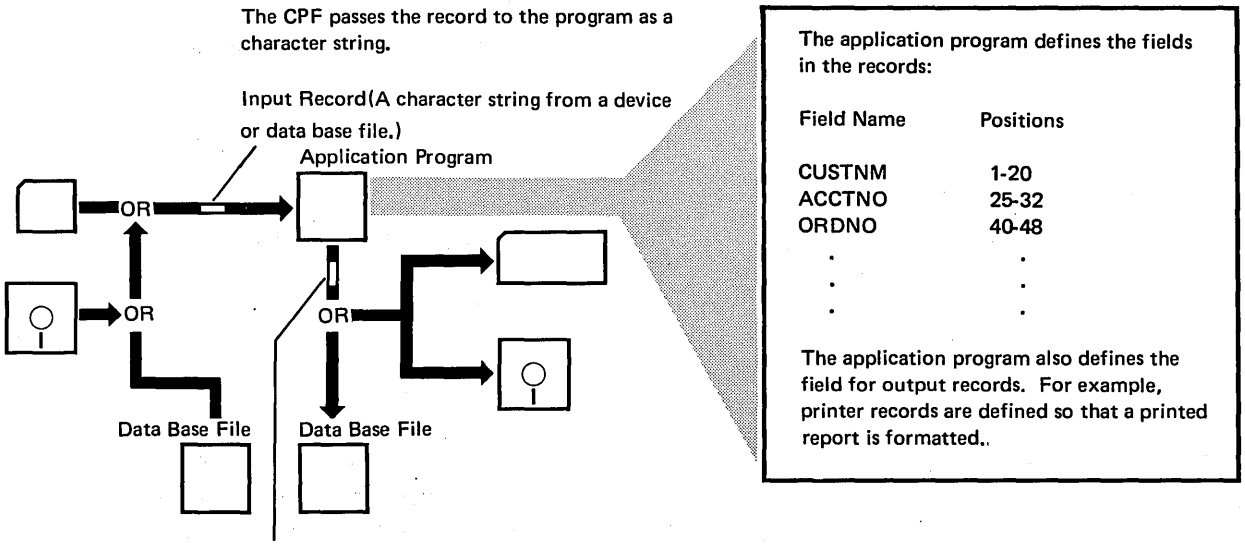
DATA OPERATIONS

The steps involved in creating a file and the requirements imposed on programs that use the file depend on whether the file is an externally described data file or a program described data file.

Program Described Data Files

CPF provides the commands needed to create, change, and delete program described data files. The same commands are used for both program described data and externally described data files. However, data description specifications are not used for program described data files. When the file is deleted, the file's description and any data in storage associated with it is destroyed.

For program described data files, the file description serves primarily as a link between the application program and the device or data used by the program. Because the data is not described to CPF, CPF treats each record as a single field containing a character string. The program using the file must identify the fields in the record by each field's location in the character string, as follows:

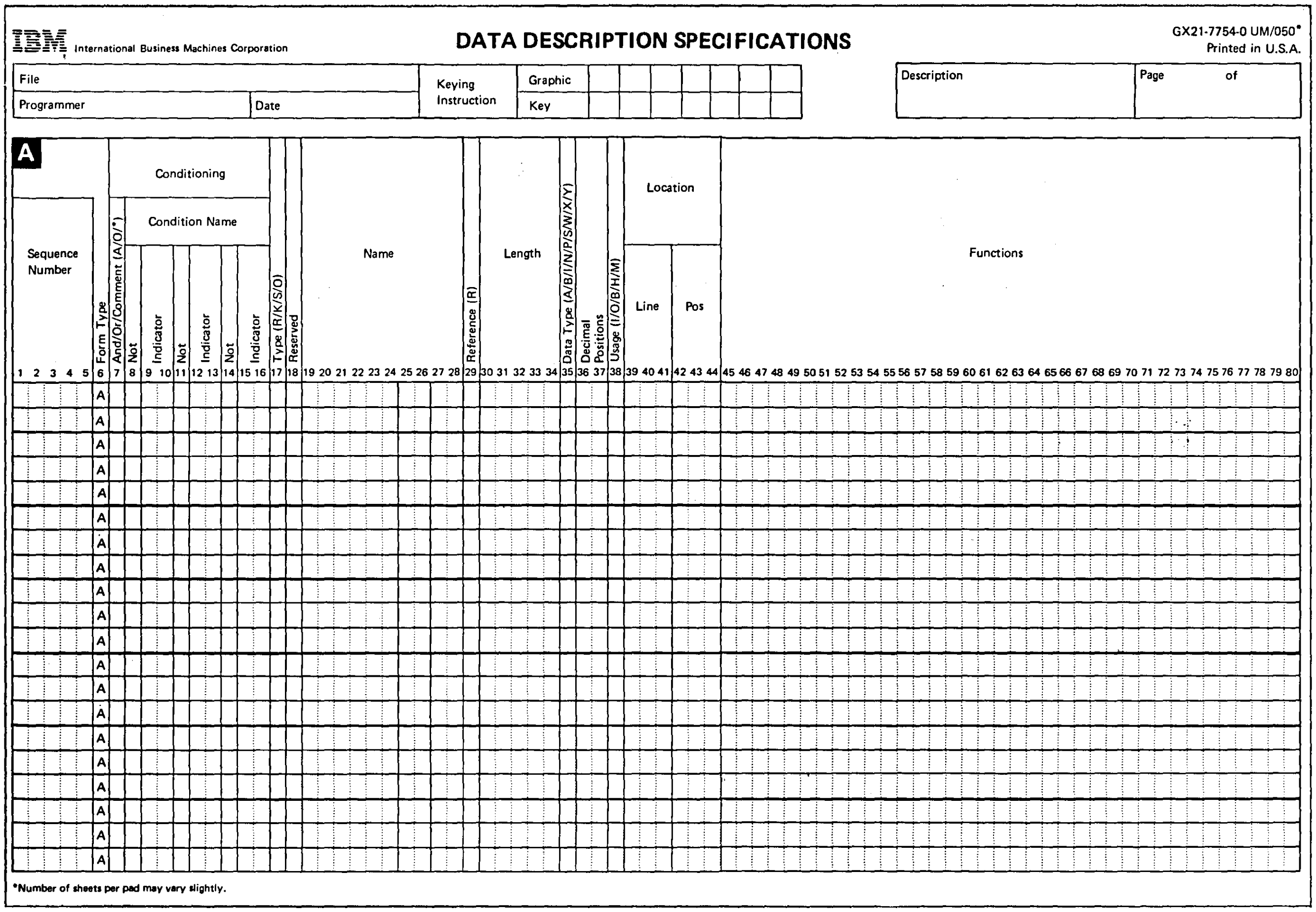


Externally Described Data Files

As with program described data files, CPF provides the commands needed to create externally described data files. However, for externally described data files the record formats used by the file must also be described to CPF by source statements provided when the file is created. The source statements are called data description specifications.

Commands are also provided to delete files and change device files. Deleting a file destroys the file's description and any data associated with it in storage.

The data description specifications form is used for coding the data description specifications for externally described data files:


IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-0 UM/050*
Printed in U.S.A.

File
Programmer
Date
Keying Instruction
Graphic
Key
Description
Page of

A

Sequence Number
Form Type
And/Or/Comment (A/O/*)
Conditioning
Condition Name
Not
Indicator
Not
Indicator
Not
Indicator
Type (R/K/S/O)
Reserved
Name
Reference (R)
Length
Data Type (A/B/I/N/P/S/W/X/Y)
Decimal Positions
Usage (I/O/B/H/M)
Location
Line
Pos
Functions

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

*Number of sheets per pad may vary slightly.

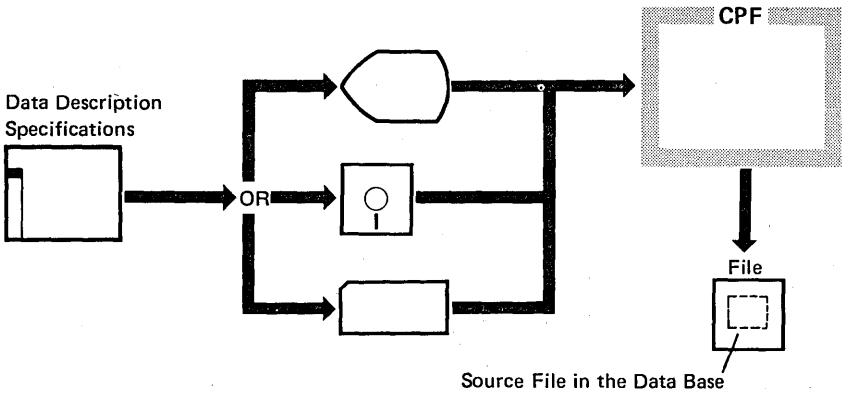
The form is used for both data base files and device files. A complete description of the data description form and the entries allowed for each position is contained in the *IBM System/38 Control Program Facility Reference Manual—Data Description Specifications*. Examples showing how the form is used are contained in the *IBM System/38 Control Program Facility Programmer's Guide*.

The data description specifications are provided as source statements when the file is created. There are two ways to enter the source statements:

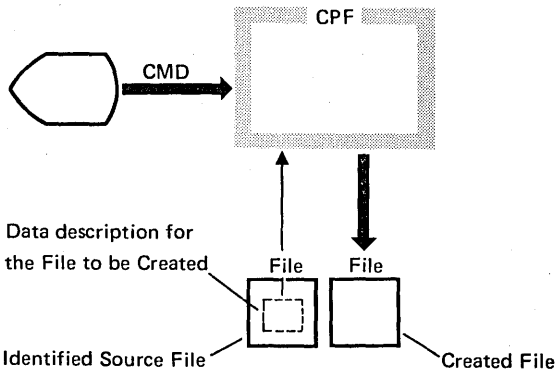
- Use a card or diskette source file
- Use the source entry utility, a part of the Interactive Data Base Utilities program product, to build a source file in the data base

The following sequence of operations is used to create an externally described data file.

1. The data description specifications are coded and entered:



2. A command is entered to create the file. This command identifies the source file that contains the data description specifications for the file:



3. The file now exists on the system.

Spooled File Processing

CPF provides spooling functions for both input and output. For input, CPF programs that are called readers read jobs and place them on a job queue. If inline data files are included with the jobs, they are placed in the system as spooled input files. For output, CPF places output records produced by a program in a spooled output file in the system. These files are later written to the external devices by CPF programs called *writers*. However, the program processes any spooled file as though the program were using the device directly.

Inline Data Files

Inline data files are processed by a program as program described data files coming from the external device. The records in the file are processed sequentially from the beginning of the file to the end of the file. Inline data files can be either unnnamed or named.

Unnamed Inline Data Files: Unnamed inline data files are identified in the spooled input by a data delimiter that does not specify a file name. Once a unnamed file has been processed by the job, it is closed and cannot be reopened. If more than one unnamed inline data file is included in a job, the files are opened in the order in which they were read in the spooled input.

Named Inline Data Files: Named inline data files are identified in the spooled input by a data delimiter that specifies a file name. Because these files are uniquely named within the program, they can be opened and processed in any order. In addition, a file can be closed and then reopened in the same job. Each time a named inline data file is opened, records are processed from the beginning of the file. The file is not deleted until the job ends.

Spooled Output Files

Output spooling functions are performed by CPF without requiring any special operations by the program that produces the files. When an output file is opened by a program, CPF determines whether the file is to be spooled. The following information in the file description applies to spooled output files:

- The output queue for the file
- The type of forms to be used
- The number of copies to be produced
- The maximum number of records that can be placed in the file
- Whether the spooled file should be written to the device while the program is still producing output
- Whether the file should be saved on the queue after it has been written to the device

This information in the file description is used when the file is opened unless the information is overridden by a control language command in the job.

When a spooled output file is opened, an entry for the file is placed on the output queue and the data is placed in a spooled file in the system. When a writer is started, the output file is selected from the queue (according to the priority assigned to the job). Spooled output is normally unavailable for output until the file is closed.

Copying Files

System/38 provides functions for copying data from one storage location or device to another. After a copy operation, the records exist in two places: in the file that was copied from, and in the file that was copied to. The copy functions can be used to copy entire files or portions of files as follows:

- Copy from any data base file to a physical data base file
- Copy from any data base file to a device file
- Copy from a device file to another device file
- Copy from a device file to a physical data base file

The copy functions also allow changes to be made to a file as it is copied. Records can be added to the receiving file, or records can completely replace any previously existing records in the receiving file.

Both externally described data files and program described data files can be copied. When externally described data files are copied, some operations, which previously could be performed only by application programs, can be requested because fields in the records have been described to CPF. These operations include:

- Selecting only a subset of records from the file being copied
- Deleting a portion of records from the file being copied
- Changing the sequence of the records so that the organization of the new file is different from that of the copied file
- Changing the format of the records as they are copied by deleting or adding fields to the record formats of the new file

File Reference Function

As application programs are developed, the programmer needs to know what data is available in the system and where it is used before he can determine whether additional files, record formats, and field definitions are needed. This information is also necessary when changes are made to the data base. CPF provides commands that can be used to determine how data is stored on the system and how it is accessed by application programs.

CPF provides facilities to track file usage on the system. These facilities simplify the work done when applications are being developed. The functions provide information about the use of both data base and device files, such as:

- Which files are used by a program; how the file is used by the program, for example input, output (or both), or update; and what record formats in the file are used by the program
- The contents of the file description, including file attributes, the record formats and access paths used for data base files, and the output queues and record formats used for a device file
- The relationships between shared files, record formats, and access paths in the data base, including which files use a specific record format or a specific access path and which logical files use the data in a specific physical file
- The individual fields included in each record format in a file, with detailed information describing each field in the record formats used by a file and secondary references to other record formats

The information generated by these facilities can be displayed at a work station or printed. In addition, most of the information can be placed in a data base file. For example, the query utility, part of the Interactive Data Base Utilities program product, could be used to examine specific information in the data base file.

OVERVIEW

The process of developing a data processing application for any system generally involves a sequence of activities such as:

1. Application design
2. Program writing
3. Application testing and debugging
4. Application documentation

Many of the functions provided by CPF are used during application development. The first part of this chapter discusses the application development activities and relates them to System/38. The rest of the chapter discusses CPF concepts related to application development that have not been presented earlier in this publication.

Design Considerations

The major design consideration is whether to implement a batch application or an interactive application. In most cases, a complete application is a combination of batch and interactive programs. For example, if an application requires documents to be printed (such as picking slips or purchase orders), that part of the application is best performed by a batch job. However, the entry of data that updates master files is needed on a timely basis and is best performed interactively. Generally, any job that takes more than a short time should be executed as a batch job so that it does not tie up the work station while the job is executed.

In System/38, batch jobs can be submitted by work station users other than the system operator. Consequently, these batch jobs can be included in the application design in such a way that the work station users can submit them whenever they are needed.

Three primary elements should be considered when an interactive application is designed:

- The interface needed by the user to invoke and communicate with the application
- The files needed for the application
- The structure of the application; that is, the programs to be included and the method used to control flow among the programs

Program Invocation: The interface used to invoke (start) an application should be designed specifically for the application user. Any program can be called through a command. However, because many work station users are not familiar with the control language, some other interface might be more convenient for those users.

CPF provides the following generalized menu (through which programs can be called), which is designed for work station users who are not familiar with the control language:

```
                    PROGRAM CALL MENU
Select one of the following:
     1.   Call program (identify below)
     2.   Display messages
     3.   Send message to system operator
     4.   Sign off work station




Option:____   Program name:_____________________________
Parameters or message:_____________________________________
_____________________________________________________________
```

To call a program using this menu, the work station user selects option 1, provides the name of the program, and enters any parameters that are required.

Application programs can also be designed so that the work station user does not need to call them. The application designer can do this by:

- Specifying that the program should be called whenever the appropriate work station user signs on the system. Through this method, the program is available to the work station user whenever he signs on.

- Specifying that the program should be invoked through a routing entry in the subsystem description. When the work station user wants to use the program, he might request it by pressing a function key or by entering the appropriate routing data. The display that requests the routing data could be a standard display supplied by CPF or a user-defined display. The routing data can be a descriptive word or phrase that describes the desired function, such as PAYROLL.

- Specifying the program as an autostart job in the appropriate subsystem description. When the subsystem is started, the job starts and allocates the appropriate work stations. The program is then ready for use by the work station user. The use of these work stations is restricted to the functions performed by the job.

Functions that are designed for users who are familiar with the control language, such as system operators or programmers, could be specifically called through IBM-supplied commands or user-defined commands. User-defined commands can be defined to provide functions that are not available through commands provided by CPF or to provide a different interface to CPF-supplied functions. CPF provides prompting for any commands defined on the system. More information on command definition is provided later under *Command Definition* in this chapter.

Program-User Communication: When an interactive program is executing, communication is needed between the program and the work station user. This communication is normally performed through a display device file.

Data Files

Designing the files used by an application is an important part of designing an application for use on any system. The CPF data management facilities provide flexibility in the use of both data base and device files.

Data Base Files: Because logical files can be used to process data stored in the data base, programs can process data using record formats and access paths that differ from those used to store the data on the system. During application design, the following decisions must be made about the use of data base files:

- What files, record formats, and access paths are needed?
- Are new physical files needed, or do the physical files already on the system contain the necessary data?
- Are new logical files necessary to provide the record formats and/or access paths needed by the program?
- Are the record formats and fields used by the program already defined in the system?

CPF provides the functions necessary to display the file descriptions, record formats, and file usage information for files that already exist on the system, as described earlier under *Data Management Facilities*.

Device Files: The device files used by a program can process either externally described data or program described data, depending on the type of device accessed by the file. The CPF functions that display file descriptions, record formats, and file usage information also apply to device files, as described earlier under *Data Management Facilities*.

Application Structure

An application can consist of a number of programs. Each program can be designed to perform a specific function. Whenever the function is needed, the program is called. When applications are structured in this manner, one program in the application controls the flow of activity within the application.

When the application is developed, each program should be written in the high-level language that best provides the functions needed. Any program can call any other program regardless of the high-level languages in which the programs are written. For example, control language programs might be needed to provide the interface through which a work station user requests application functions. Control language programs also can be used to call other programs based on conditions that exist during program execution.

The high-level languages provide the facilities needed to perform operations on the data processed by the application. These programs can request data base manipulation functions through the data management functions provided by the CPF. Because control language programs do not perform the processing normally associated with data base files, these functions are not available through control language programs.

Applications that are made up of more than one program must provide a mechanism through which information can be passed from one program to another. Information can be passed:

- In the parameters of the command that invokes the program
- In messages supported by the CPF message handling facilities (described later under *Message Handling* in this chapter)
- In a data area object contained in a library on the system

A *data area* object contains common data that can be shared by different programs in a job or by programs in different jobs. It exists independently of the programs that use it. Values can be placed in the data area to control the functions performed by programs that access those values. The values can be changed by the programs or by commands entered by a work station user. A facility is provided to synchronize the use of values in the data area so that one program does not change a value while another program is using it.

Programming Considerations

After an application has been designed, the files used in the application must be described and created and the various programs must be coded and compiled. Data description specifications are source statements for externally described data files. Control language commands and other high-level language source statements must be provided for programs that will be created.

Entering Source Statements

For programs or externally described data files to be created, source files containing the source statements must be provided. The user can place a source file in the data base by copying the file from a device file or by entering the source statements through the source entry utility, which is part of the Interactive Data Base Utilities program product. A source file can also be either a spooled or nonspooled device file that is provided when the program or file is created.

The source entry utility provides prompts, to help the work station user enter commands, and it provides special display formats to help the user enter specifications for other programs and data description. The utility also can perform syntax checking on the source statements that are entered. A complete description of the source entry utility is contained in the *IBM System/38 Source Entry Utility Reference Manual and User's Guide.*

Creating Programs and Files

CPF provides the commands that are needed to create files and programs from the source statements contained in source files. These commands can be used in either interactive or batch jobs.

Any externally described data files must be created before the programs that use them because the record formats from the file descriptions are copied into the program when it is created. A program is compiled when it is created and then exists as a program object that can be called to be executed.

Testing and Debugging

Application programs that manipulate data stored in the system are normally tested with sample data before they are executed using normal production data files. CPF provides the functions needed to test applications against sample data in a protected environment and also provides functions that help a programmer debug his programs.

Programs that run in the protected environment can use data from files in any library, but they can update only those files contained in test libraries. Thus data files that are used in normal data processing operations are protected from unintentional modification by the program being tested.

This environment can also be a useful tool for protecting production data files while work station users are being trained to use the program. Because the program executes the same way in either environment, the work station user can be fully trained before he uses the normal operating environment.

If errors occur in the functions performed by a program when it is tested, the cause of these errors must be detected. The debugging facilities provided by CPF can be used in the protected environment for debugging any high-level language program, including control language programs. These functions do not require any special coding within the program. Specific points in the program are identified by the labels and statement identifiers used in the program's source statements. More information on the CPF debugging functions is provided under *Debugging Functions* in this chapter.

After a program has been tested and debugged, the source for the program can be changed to correct any errors found during testing. When the source has been updated, the program can be created again (recompiled) and retested.

Documentation

Good documentation is a key element in the maintainability of any data processing system or application program. CPF allows up-to-date documentation to be maintained on the system itself. In addition to the use of comments, which is supported by most high-level languages, CPF provides:

- The ability to include a text description of any object (including libraries) in the object description on the system. The text description can be provided when the object is created and changed through commands that change the object. The text description is displayed when the object description is displayed.

- The ability to include a text description in the data description specifications for any record format or for individual fields within the record format. These text descriptions are stored in the file description and are displayed when the file description is displayed. They also become comments in all the programs that use externally described data, which provides complete and consistent program documentation of the data to the field level.

- The ability to use CPF functions to provide a cross-reference of the use of files and record formats by programs.

Text descriptions and comments can be included in and stored with the program source statements, but they are not used as input to a compiler.

CONTROL LANGUAGE PROGRAMS

A *control language program* is made up of control language commands. The commands are compiled into an executable program that can be called whenever the functions provided by the program are needed. The use of control language programs in an application can provide many advantages. For example:

- Because the commands are compiled and stored in executable form, using control language programs is faster than entering and executing the commands individually.
- Certain functions that are not available when commands are entered individually are available in control language programs.
- Control language programs can be tested and debugged like other high-level language programs.
- Parameters can be passed to control language programs to adapt the operations performed by the program to the particular requirements of that use.

Control language programs can be used for many kinds of applications. For example, control language programs can be used to:

- Provide an interface to the user of an interactive application through which the user can request application functions without an understanding of the commands used in the program. This makes the work station user's job easier and reduces the chances of errors occurring when commands are entered.
- Control the operation of a batch application by establishing variables used in the application (such as date, time, and external indicators) and specifying the library list used by the application. This ensures that these operations are performed whenever the application program is executed.
- Provide predefined procedures for the system operator, such as procedures to start a subsystem, to provide backup copies of files, or to perform any other procedural operating functions. The use of control language programs to perform these procedures reduces the number of commands the operator uses regularly, and it ensures that system operations are performed consistently.

Most of the control language commands provided by CPF can be used in control language programs. In addition, some functions designed for use in control language programs are not available when commands are entered individually. These functions include:

- Logic control functions that can be used to control which operations are performed by the program according to conditions that exist when the program is executed. For example, *if* a certain condition exists, *then do* certain processing, *else* (otherwise) do some other operation. These logic operations provide both conditional and unconditional branching within the control language program.

- Data operations that provide a way for the program to communicate with a work station user. These operations let the program send formatted data to the work station and receive data from the work station.

- Functions that allow the program to send messages to, and receive messages from, the work station user.

- Functions that receive messages sent by other programs. These messages can provide normal communication between programs or indicate that errors or other exceptional conditions exist.

- The use of variables and parameters for passing information betweem commands in the program and between programs.

Through the use of control language programs, applications can be designed with a separate program for each function and with a control language program controlling the execution of all the programs within the application. The application can consist of control language programs and other high-level language programs. In this type of application, control language programs are used to:

- Determine which programs in the application are to be executed

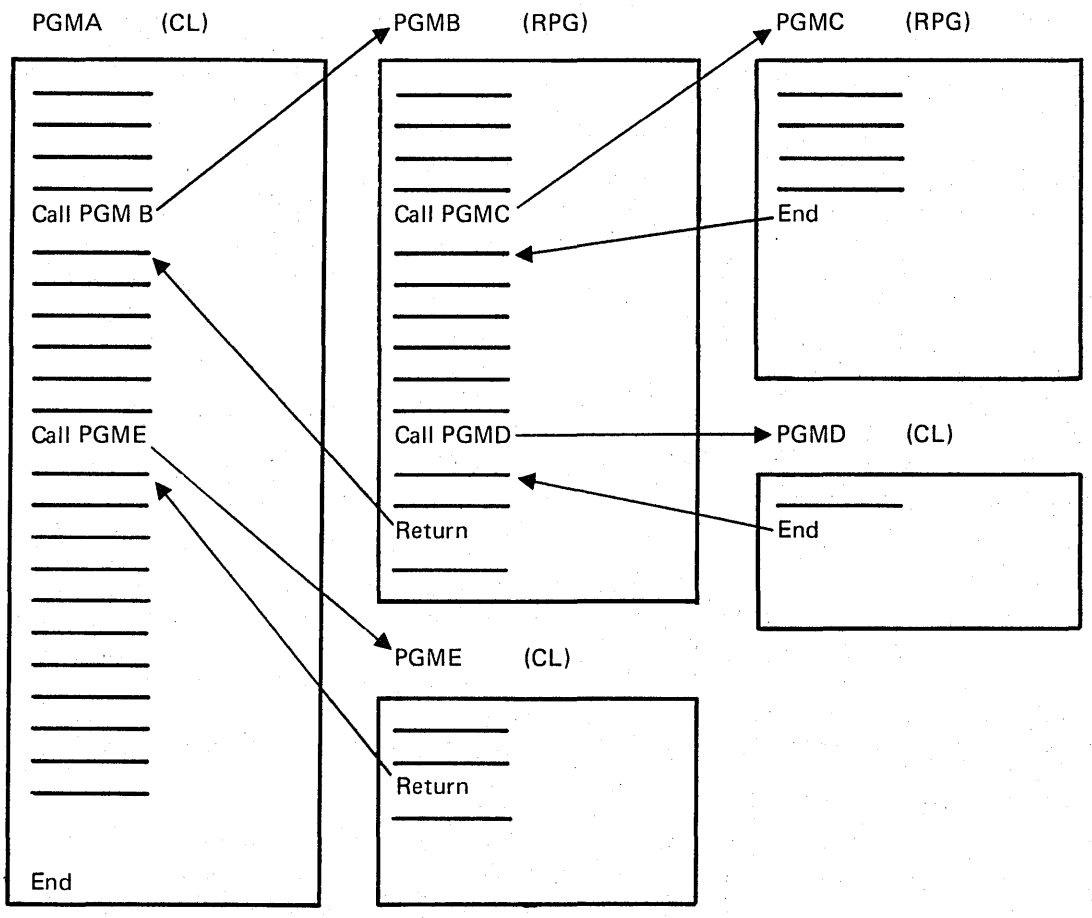
- Provide system functions that are not available through other high-level languages

- Provide interaction with the application user

Thus control language programs provide the flexibility needed to let the application user select the operations he wants to perform and to execute the necessary programs.

The following example shows how control could be passed between a control language program and RPG programs in an application. To use the application, a work station user would request program A. Program A controls the entire application. The example shows:

- A control language program calling another control language program
- A control language program calling an RPG program
- An RPG program calling another RPG program
- An RPG program calling a control language program



MESSAGE HANDLING

A *message* is a communication sent from one person or program to another. CPF provides message handling functions that can be used to communicate between programs and system users.

The following concepts of message handling are important to application development:

- Messages can be defined outside the programs that use them in message files, and variable information can be provided in the message text when a message is sent. Because messages are defined outside the programs, the programs do not have to be changed when the messages are modified.

- Messages are sent to and received from message queues, which are separate objects on the system. A message sent to a queue can remain on the queue until it is explicitly received by a program or work station user.

- A program can communicate with the work station user who requested the program without the messages being sent to a specific device by the program. Thus one program can be used from different work stations without change.

Because replies can be returned by a user or program that receives a message, the message handling facilities provide a mechanism for two-way communication.

Message Descriptions

A *message description* defines a message to CPF. In addition to information about the message, the description contains the text of the message. This message text can include variable data that is provided by the message sender when the message is sent.

Message descriptions are stored in message files. Each description must have an identifier that is unique within the file. When a message is sent, the message file and message identifier specify to the CPF the message description that is to be used.

CPF supports message types that allow many kinds of messages. Through these message types, information, inquiries, requests and replies can be sent between users and programs. In addition, completion messages and various types of diagnostic messages are supported to provide information about the status of work on the system.

Message Queues

When a message is sent to a program or a system user, it is placed on a *message queue* associated with that program or user. The program or user obtains the message by receiving it from the queue. Thus, the message does not need to be processed immediately when it is sent.

CPF provides message queues for:

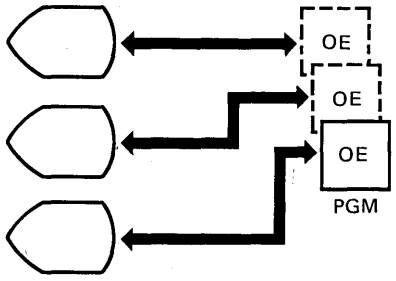
- Each work station on the system
- Each job on the system and each active program within a job
- The system operator
- The system logs

Additional message queues can be created to meet any special application requirements. Messages are actually sent to message queues, where they are retained, so the receiver of the message does not need to process the message immediately. Thus a message queue can be used as a mailbox to hold the messages until the appropriate program or user receives the message.

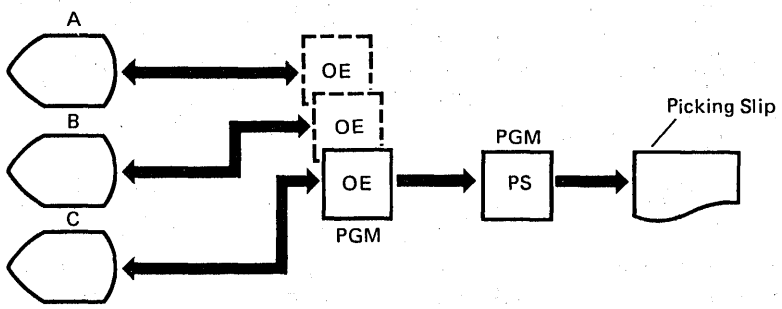
Using Messages and Message Queues

Messages and message queues can be used both to pass information and to request processing by programs in an application. For example, an application used for entering orders into the system could use messages and message queues as follows:

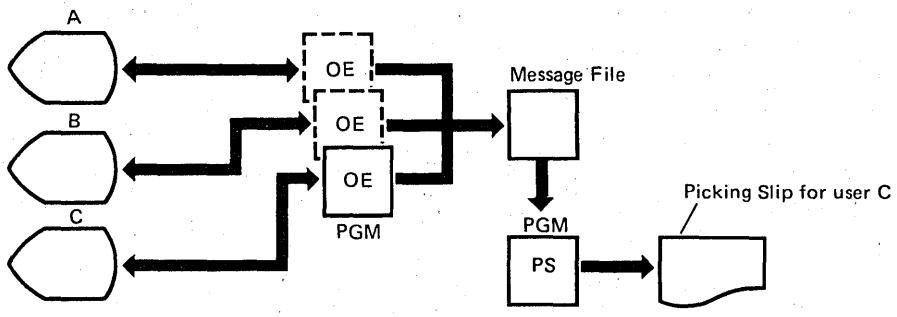
1. Three work station users enter orders using the same order entry program. (Because of the design of the system, all three users actually share a single copy of the executable program.)



2. Once an order has been entered, the application must produce a picking slip needed for filling the order. One program is used for producing picking slips. Because this program interacts directly with the printer, if it were called from one order entry program, it could handle orders from only one user at a time, thus delaying the work station users while the printer is in use. The following drawing shows this type of operation. User C has completed an order, so program PS has been called to print the picking slip. Program PS is now unavailable to the other users.



To prevent this delay, the program that produces picking slips processes entries by processing messages from a message queue, as shown in the next drawing. The order entry program sends a message to that queue for each order and then continues processing.



DEBUGGING FUNCTIONS

CPF includes functions that let a programmer interact with a program as it executes to observe the operations being performed. These debugging functions can be used when the program is executing in the testing environment.

The debugging functions narrow the search for errors that are difficult to find in the program's source statements. Often, an error is apparent only because the output produced is not what is expected. To find those errors, a programmer needs to be able to stop the program at a given point and examine variable information in the program to see if it is correct. He might want to make changes to those variables before letting the program continue executing. The programmer does not need to know machine language instructions, nor does he need to include special instructions in the program to use the debugging functions. The CPF debugging functions let the programmer:

- Stop the execution of the program at any named point in the program's source statements.
- Display the variable information used by the program as it exists when program execution is stopped. If he wants to, the programmer can also change the variable information before program execution is resumed.
- Trace the use of variables in the program by recording the steps in the program that change the variables and what those changes are. This operation produces a printout or display that traces the execution sequence, showing which statements in the program are executing and what the value of a variable is at any point in the program.

COMMAND DEFINITION

Through the control language commands, the support of control language programs, and other CPF functions, CPF provides the functions normally needed for developing application programs. However, advanced uses of the system might require redefinition of some control language commands or the creation of additional commands to meet the specific needs of an installation. CPF includes functions that allow the creation of user-defined commands.

Each command on the system has a *command definition* object and a command processing program. The command definition object defines the command, including:

- The command name
- The command processing program
- The parameters and values that are valid for the command
- Validity checking information that CPF can use to validate the command when it is entered
- Prompt text to be displayed if a prompt is requested for the command

The command processing program is the program that CPF calls when the command is entered. Because CPF performs validity checking when the command is entered, the command processing program does not have to check the parameters passed to it.

The command definition functions can be used to:

- Create unique commands needed by an installation.
- Redefine commands provided by CPF to meet the requirements of an installation. This might include changing the defaults for parameter values or simplifying the commands so that some parameters would not need to be entered. Constant values can be defined for those parameters.
- Create specialized commands to meet the needs of individual users.

More detailed information on command definition is contained in the *IBM System/38 Control Program Facility Programmer's Guide* and *IBM System/38 Control Program Facility Reference Manual.*

The System/38 CPF provides the facilities that are needed to regulate the use and operation of a data processing installation. These facilities are designed to augment the facilities described in previous chapters to provide system-wide management and control. System management includes:

- Controlling the use of system resources
- Backing up the system and objects in the system
- Installing new support
- Operating the system
- Servicing the system

SECURITY

In an interactive system, the implementation of controls that ensure data integrity and security becomes especially important because the work stations provide many points of direct access to the system outside the physical control of the data processing department. Without these controls, the potential for data being misused or destroyed increases, especially when many work station users are using the system concurrently.

For many data processing installations, maintaining the security and integrity of data processing information is a primary concern. The most important concern is *integrity*: the protection of programs and data from inadvertent destruction or alteration. *Security* is the prevention of access to or use of data or programs by unauthorized persons. Directly related to integrity and security is the need for *user identification*: the ability to recognize a system user so that only the facilities and data he is authorized to use are made available to him.

The security facilities of CPF provide mechanisms for user identification and for authorizing user access to specific objects. These facilities allow the system to be tailored to provide the necessary level of security and integrity. In addition, the user identification supported by these facilities can be used to design an application-oriented interface for work station users. The system can be tailored so that each work station user has access to only the system functions, applications, and data that he needs to perform his work.

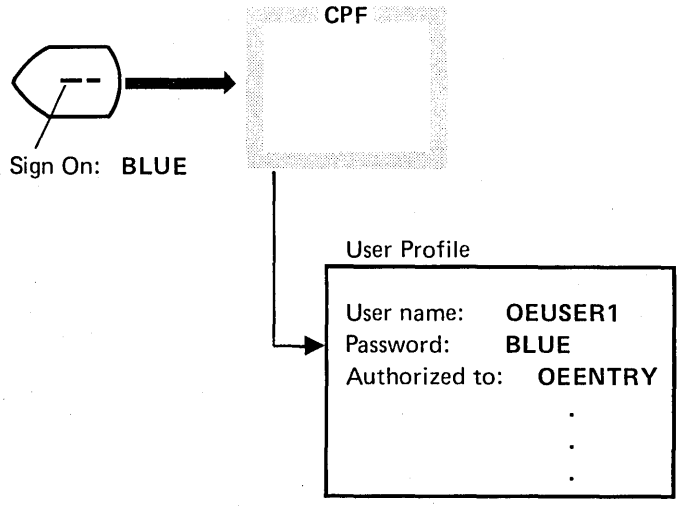
User Identification

All System/38 security functions rely on a user profile to identify each system user. A *user profile* is an object that represents a particular user or group of users to CPF. The user profile identifies which objects and functions the user is authorized to use. When a work station user signs on to the system, he enters a *password* to identify himself. CPF uses that password to determine which user profile represents that user. If more than one user signs on the system using the same password, they are represented to CPF by the same user profile and have access to the same objects and functions. A password is usually known only by the person or persons who use it. To help prevent unauthorized use, passwords can be changed as desired.

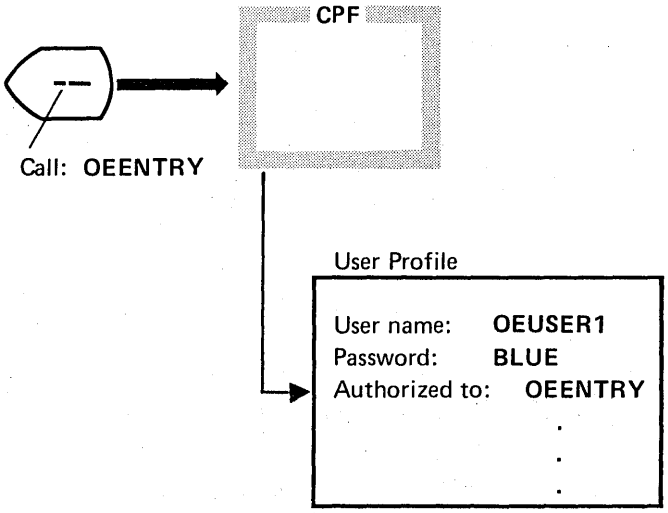
When a work station user signs on, CPF uses the password he enters to find the user's profile. Each user profile is named. The user is known in the system by the name of his user profile. Thus references to a user in the system do not need to be changed when a password is changed. The following drawing shows how the password and the user name are used by CPF.

Work station user (OEUSER1) signs on using his password. The password (BLUE) is not displayed when it is entered.

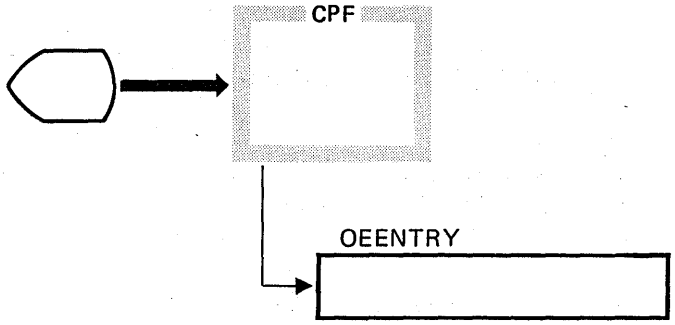
1. The CPF uses the password (BLUE) to determine which user profile to use.



2. The work station user requests the program OEENTRY. The CPF determines whether the user profile (OEUSER1) is authorized to use the object.



3. If the user is authorized to the object, the request is honored.

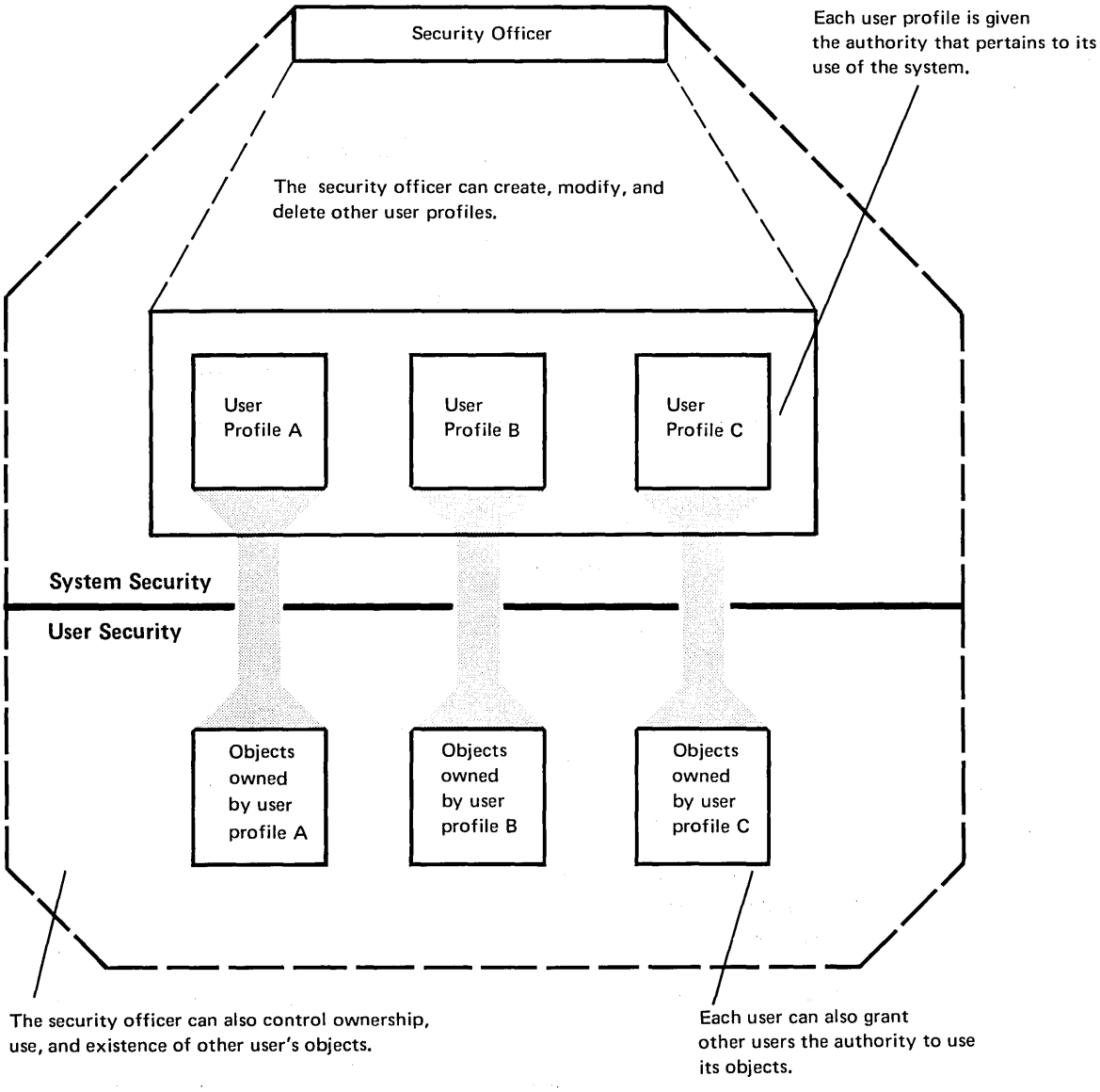


When CPF is installed, it includes a set of predefined user profiles. Together, these profiles allow the use of all the system functions. The predefined user profiles are:

- The system security officer–lets one system user control user profiles and other security functions.
- The programmer–lets a system user perform the functions necessary to develop system or application programs.
- The system operator–lets a system user perform the functions necessary to operate the system.
- The work station user–lets a system user operate work stations.
- The program support representative–lets service personnel maintain CPF.
- The customer engineer–lets service personnel use the concurrent service monitor to maintain the machine.

Security Functions

Because all the functions and data available on the system exist as objects, their use is controlled by the specific authorization of system users. As shown in the following drawing, CPF provides two levels of security functions: system security and user security.



System Security Functions

The system security functions require the use of the system security officer user profile. The system security officer can:

- Enroll users on the system by assigning them to a user profile that is provided by CPF or that he has created
- Grant or revoke the authority for a user to use specific system functions, subsystems, application programs and any other objects on the system
- Revoke a user's authority to sign on the system by changing the password or deleting the user profile from the system

Thus the system security officer has the ultimate control over the use of the system.

User Security Functions

Within the limits established by the user profile, each system user can control the use of his objects. User security functions are provided through object authorization.

Object Authorization

Object authorization is the process of controlling which system users are allowed *(authorized)* to use an object and how each user can use the object. Two basic concepts are involved in object authorization: object ownership and object authority.

Object Ownership

Whenever a system user creates an object, he becomes the owner of that object. Unless ownership is transferred to a different user, he remains the owner of the object until the object is deleted from the system. The owner has complete control over his object. He can authorize other system users to use the object and he can transfer ownership of the object to some other system user. Only the system's security officer has the same control over an object as the object's owner.

Object Authority

Each system user must be authorized to use each object he needs. When an object is created, three levels of public authority to use the object can be established:

- All authority, which allows any operation involving the object by all system users.
- No authority, which allows no operations by anyone except the object owner.
- Normal authority, which allows the operations normally associated with the object to be performed by all system users. For example, the normal operation for a program would be its execution.

After the object is created, any authority can be granted or revoked for specific users or for the public.

How a user can use an object depends on what rights of object use are included in his authority. The object's owner and the system's security officer always have all rights to the use of an object. Other system users can be granted some or all rights of object use either through public authority or explicitly granted authority. The authority that can be granted is in two categories:

- *Object rights* control what the user can do to the entire object. For example, object rights can let a user delete, move, or rename an object.
- *Data rights* control how the user can use data in the object. For example, the data rights might let a user read and update data in a file. Data rights provide additional control over the use of data entries within objects and are granted in addition to the object rights a user has.

Using Security

An installation's security needs should be considered whenever application programs are designed. Application programs can be designed so that the security can be increased, as required by future needs, without unnecessary changes to the application programs. The following are typical security considerations:

- Each system user should have access only to the functions and data he needs to perform his job.
- Work station users should be able to access and update data in the data base only through thoroughly tested programs.

A user can be authorized to use the objects he needs through the program he uses. To use this function, programs are created that execute with the user profile of the program's owner in addition to that of the user who calls the program. As long as the user is using the program, he has access to the objects and functions used by the program. This kind of operation offers such advantages as the following:

- System users are authorized to use the objects without requiring numerous explicit authorizations to be made.
- Additional work station users can easily be authorized to use the application.
- Changes and additions can be made to the application without requiring additional explicit authorizations.
- Security can be established through the same program that provides the interface to the user.

The use of this function ensures that all users have access to the system functions they need without public authority being granted for many objects. As with any other security functions, the program should be designed to prevent users from circumventing the controls that are established.

SAVE/RESTORE

CPF includes the functions needed to save objects offline and later restore them to the system. These save/restore facilties can be used to establish the procedures to be used to back up the system. These procedures can be designed as an integral part of system operations. The save/restore facilities can also be used to save seldom-used objects and free their auxiliary storage for other objects, and to store sensitive objects offline in a physically secure location to prevent access by unauthorized persons. CPF provides functions to:

- Save objects from the system by writing a copy of the objects to offline storage and, optionally, free the auxiliary storage that is occupied by the objects so the space can be used for other objects
- Restore saved objects to the system

These functions can be used to create backup copies of entire libraries or of individual objects on the system. The use of save/restore functions is important in maintaining a system that can recover from failures quickly and easily.

Save Functions

The save functions write a copy of an object onto diskettes. The object's description is also saved. The object is not removed from the system when it is saved; it still exists in the system and is available for normal use. In a single operation, a copy of a single library, of a group of libraries, of a single object, or of a group of objects in one library can be saved.

CPF maintains save/restore history information about each object saved. The information tells CPF when and where each object was saved and when the object was last restored. The information always applies to the most recent save/restore operations for each object. The information is used to ensure that objects are not inadvertently restored from an outdated copy of the object. This information can be displayed through the use of control language commands.

If you want to make the storage occupied by the data portion of an object available for other system use, you can also free the object's storage when you save the object. After the storage has been freed, the object is *offline*. When an object is offline, its description and offline location are still maintained in the system. However, space from the contents of the object is freed. Thus some operations, such as displaying the object description, can still be performed. Freeing the object's storage is not the same as deleting an object. When an object is deleted, all information about it is removed from the system; the object must be created or restored before it can be used again.

Restore Functions

The restore functions of CPF copy saved copies of objects back into the system. These functions are used to restore any saved object except those in the system library. These objects are copied back into the system by the CPF installation and specialization facilities described in this chapter.

Using Save/Restore

System operating procedures should include a plan for backing up the system to recover from system or application failures. The plan should identify which objects must be saved and how often the save operations must be done.

All the objects in a system do not all need to be saved at the same time. Some objects are used often but are seldom changed, and so they might be saved only after a change has occurred. Other objects might contain crucial information that changes daily. These objects might be saved daily to provide an up-to-date copy of the object for backup purposes.

The ability to recover from errors and application failures should be part of the design of an application. When an application is designed, its design should include the approach to be used to maintain a backup copy of the files and programs used by the application.

INSTALLATION AND SPECIALIZATION FACILITIES

CPF includes the facilities needed to perform:

- Initial installation of the IBM-provided objects that make up CPF
- Installation of IBM-provided objects that are distributed as updates or enhancements to a previously installed CPF
- Installation of CPF libraries that were saved by the save/restore facilities

After CPF is installed, System/38 is operational and can be used to satisfy many data processing requirements. CPF can be tailored (specialized) to meet specific data processing requirements. Specialization of CPF, which is performed with control language commands, can be performed at any time after CPF is installed. Specialization might include:

- Defining the lines, control units, and work stations on the system to CPF
- Creating any unique print images and edit codes needed by the application programs
- Creating user-defined libraries
- Creating any additional subsystem descriptions, job descriptions, job queues, and classes that are needed to manage the work done on the system
- Creating any additional message queues needed by applications run on the system
- Creating additional output queues to be used by the spooling functions of CPF
- Creating any additional user profiles and authorizing the various system users to use the objects they need

Control language commands are also used to install other program products.

SYSTEM OPERATION

The operation of System/38 is controlled through control language commands and system messages. The system operator uses commands to control and terminate the system, subsystems, and other functions on the system. He can also use the message handling facilities to monitor the operation of the system. Although system operation is normally controlled through the system console, once the system is started, the operator can sign on at any work station and perform his normal system operation functions. The system console can also be used as a normal work station. In addition, System/38 and CPF are designed for semiattended operation. Once the system is started and the devices are ready for operation, the system can operate with minimal operator attention.

System Operation Functions

When CPF is installed, it includes a user profile for the system operator. This user profile authorizes the system operator to use the CPF functions and objects that are normally needed to operate the system. The operations associated with system operation include:

- Starting the system
- Starting and controlling the operation of subsystems
- Controlling input/output devices when intervention is required
- Controlling spooling functions
- Performing save/restore operations
- Handling diagnostic messages that CPF provides indicating errors or exceptional conditions that have occurred

Message Handling

When CPF is installed, it includes a message queue for the system operator. Messages from CPF intended for the system operator are sent to this message queue. Work station users and application programs can also send messages to this message queue. The system operator can receive messages from this queue regardless of the work station at which he signs on.

The system operator and other work station users can communicate with each other by using the message handling facilities. Two types of messages are supported by CPF:

- Predefined messages, which are created before they are used. These messages are placed in a message file when they are created and are retrieved from that file when they are used.
- Impromptu messages, which are created when they are sent and are not permanently stored in the system.

A message can be sent to a specific work station user (actually, to the queue associated with his work station) or to all the work stations on the system. For example, if the system operator needs to inform all the work station users that he is terminating the system, he can send one message to all the work stations in one operation.

SERVICE

CPF lets service personnel perform most service functions concurrently with normal data processing operations. The service facilities of CPF provide support for handling CPF problems, work station problems, and machine problems. The following support is provided to handle CPF problems:

- Analyzing and diagnosing the problem. Commands are provided that produce diagnostic information. Dumps of specific objects, dumps of internal job information, and traces of processing flow can be obtained through these commands. A dump is automatically generated if a job terminates because of an unexpected exceptional condition.

- Reporting problems to IBM. A command is provided to copy previously produced diagnostic information onto a diskette so that it can be submitted to IBM as documentation of a problem.

- Installing program patches and changes. Commands are provided so that temporary repairs (patches) can be applied to a program. Commands are also provided to apply IBM-supplied changes.

CPF provides commands to perform the following functions to service internal machine problems:

- Copy the contents of the machine error log to a spooled printer file

- Produce a trace of the internal machine activities

- Start the machine problem determination procedures

CPF also provides support through which work station device operation can be checked and work station printer operation can be verified.

access path: The means by which the Control Program Facility provides a logical organization to the data in a data base file so that the data can be processed by a program. See also *arrival sequence access path* and *keyed sequence access path*.

activity level: An attribute of a storage pool that specifies the maximum number of jobs that can execute concurrently in the storage pool.

arrival sequence access path: An access path that is based on the order in which records are stored in a physical file.

authority: The right to access objects, resources, or functions.

autostart job: A job that is automatically initiated when a subsystem is started. Autostart jobs are specified for a subsystem by autostart job entries in the subsystem description.

batch job: A group of processing actions submitted as a predefined series of actions to be performed without a dialog between the user and the system.

class: An object that specifies the execution parameters for a routing step. The class object is specified in the routing entry in a subsystem description.

command: A statement used to request a function of the system. A command consists of a command name, which identifies the requested function, and parameters.

command definition: An object that defines a command (including the command name, parameters, and validity checking information) and identifies the program that performs the function requested by the command.

control language: The set of all commands with which a user requests functions of the Control Program Facility.

control language program: An executable object that is created from source consisting entirely of control language commands.

controlling subsystem: The interactive subsystem that is started automatically when the system is started and through which the system operator controls the system.

data base: The collection of all the files stored in the system. Files in the data base are called data base files. See also physical file and logical file.

data description specifications: A description of data base or device files entered using a fixed-form syntax. The description is used to create the files.

data rights: The authority that controls how a system user can use the data contained in an object.

device description: An object that contains information describing a particular device that is attached to the system.

device file: A file that is processed on an external input or output device attached to the system, such as a work station, a card read and punch unit, a printer, or the diskette magazine drive.

externally described data: Data contained in a file for which the fields in the records are described to the Control Program Facility, through the use of the data description specifications, when the file is created. The field descriptions can be used by the program when the file is processed.

file description: The information contained in the file that describes the file and its contents. The data in the file can be described to the record level (see *program described data*) or to the field level (see *externally described data*).

file overrides: Parameters that are specified when a file is used to temporarily change parameters specified when the file was created.

general-purpose library: The library provided by the Control Program Facility to contain user-oriented, IBM-provided objects and user-created objects that are not explicitly placed in a different library when they are created.

inline data file: A data file that is included as part of a job when the job is read from an input device by a reader program.

integrity: The protection of data and programs from inadvertent destruction or alteration.

interactive job: A job in which the processing actions are performed in response to input provided by a work station user. During the job, a dialog exists between the user and the system.

job: A single identifiable sequence of processing actions that represents a single use of the system. A job is the basic unit by which work is identified on the system.

job description: An object in which the attributes of a job can be predefined and stored.

job queue: A queue on which batch jobs are placed when they are submitted to the system and from which they are selected for execution by the Control Program Facility.

keyed sequence access path: An access path that is based on the contents of key fields contained in the records.

key field: A field, contained in every record in the file, whose contents are used to sequence the records when the file is used.

library: An object that serves as a directory to other objects. A library is used to group related objects and to find objects by name when they are used.

library list: An ordered list of library names indicating which libraries are to be searched, and the order in which they are searched, to find an object.

logical file: A data base file through which data that is stored in one or more physical files can be accessed by means of record formats and/or access paths that are different from the physical representation of the data in the data base.

members: An identifiable group of records that is a subset of the data base file to which it belongs. Each member conforms to the characteristics of the file and has its own access path.

message: A communication sent from one person or program to another.

message description: A definition of a message that provides descriptive information about the message and contains the text of the message.

message queue: A queue (associated with a person or program) on which messages are placed when they are sent to the person or program. The person or program obtains the message by receiving it from the message queue.

object: A named unit that consists of a set of attributes (that describe the object) and data. The term object is used to refer to anything that exists in and occupies space in storage on which operations can be performed. Some examples of objects are programs, files, and libraries.

object rights: The authority that controls what a system user can do to an entire object. For example, object rights can let a user delete, move, or rename an object.

password: A unique string of characters that a system user enters to identify himself to the system.

physical file: A data base file that contains data records. All the records have the same format. That is, a physical file contains fixed-length records, all of which contain the same fields.

program described data: Data contained in a file for which the fields in the records are not described through the Control Program Facility. The fields must be described in the program that processes the file.

qualified name: The combination of an object name and a library name used to identify an object.

reader: A Control Program Facility program that reads jobs from an input device and places them on a job queue.

routing data: A character string that the Control Program Facility compares with character strings in the routing entries to select the routing entry to be initiated for a routing step. Routing data can be provided by a work station user, specified in a command, or provided through the work entry for the job.

routing entry: An entry in a subsystem description that specifies the program to be invoked to control jobs that execute in the subsystem.

routing step: The processing performed as a result of invoking a program specified in a routing entry.

security: The prevention of access to or use of data or programs by unauthorized persons.

source file: A file created to contain source statements for such items as high-level language programs and data description specifications.

spooled file: A device file that is not intended for direct access to a device but that provide access to data processed by the readers and writers.

storage pool: A quantity of main storage available for use by jobs executing in the storage pool. The storage pool does not consist of a given block of storage; rather it specifies an amount of storage that can be used.

subsystem: A predefined operating environment through which the Control Program Facility coordinates work flow and resource usage.

subsystem attributes: Specifications in a subsystem description that specify the amount of main storage available to the subsystem and the number of jobs that can execute concurrently in the subsystem.

subsystem description: An object that contains the specifications that define a subsystem and that the Control Program Facility uses to control the subsystem.

system library: The library provided by the Control Program Facility to contain system-oriented objects provided as part of the Control Program Facility.

temporary library: A library that is automatically created for each job to contain objects created by that job that are not specifically placed in another library. The objects in the temporary library are deleted when the job ends.

user identification: The ability to recognize a system user so that only the facilities and data he is authorized to use are made available to him.

user profile: An object that represents a particular user or group of users to the Control Program Facility. The user profile identifies which objects and functions the user is authorized to use.

work entry: An entry in a subsystem description that specifies a source from which jobs can be accepted to be executed in the subsystem.

writer: A Control Program Facility program that writes spooled output files from an output queue to an external device, such as a printer.

READER'S COMMENT FORM

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

Note: All comments and suggestions become the property of IBM.

Name _____

Address _____

● No postage necessary if mailed in the U.S.A.

IBM System/38
Control Program Facility
Concepts

GC21-7729-0

Cut Along Line

IBM System/38 Control Program Facility Concepts Manual

Fold

Fold

FIRST CLASS
PERMIT NO. 40
ARMONK, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM System/38 Control Program Facility Concepts Manual (File No. S/38-36) Printed in U.S.A. GC21-7729-0