GC21-7729-2

File No. S38-36

IBM System/38

IBM System/38
Control Program Facility Concepts Manual

Program Number 5714-SS1

GC21-7729-2

File No. S38-36

IBM System/38

IBM System/38
Control Program Facility Concepts Manual

Program Number 5714-SS1

Third Edition (February 1981)

This is a major revision of, and obsoletes, GC21-7729-1 and technical newsletter GN21-8130. This edition applies to release 2 of the IBM System/38 Control Program Facility (Program 5714-SS1); and to all subsequent releases until otherwise indicated in technical newsletters or in new editions. Changes are periodically made to the information herein; these changes will be reported in technical newsletters or in new editions of this publication. Changes and additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

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.

© Copyright International Business Machines Corporation 1978, 1979, 1981

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.

Note: This publication follows the convention that *he* means *he or she.*

Prerequisite Publications

- *IBM System/38 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,* SC21-7731. This publication explains the control language syntax.
- *IBM System/38 Control Program Facility Reference Manual—Data Description Specifications,* SC21-7806. This publication explains the specifications form used for describing data.
- *IBM System/38 Control Program Facility Programmer's Guide,* SC21-7730. 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 Glossary and Master Index,* GC21-7727. This publication contains a glossary of all terms used in System/38 customer publications, and combines index entries from the indexes of frequently used System/38 publications.
- *IBM System/38 Source Entry Utility Reference Manual and User's Guide,* SC21-7722. This publication describes how to use SEU to create and update source records.

Contents

CHAPTER 1. INTRODUCTION 1-1
CPF Overview 1-2
Object Management 1-2
Work Management 1-3
Data Management 1-3
Application Development 1-4
System Management 1-4
System Concepts Overview 1-5
Object-Oriented Architecture 1-6
Single Level Storage 1-6
Activity Level Control 1-8
Access Groups 1-8
Data Base Functions 1-8
Control Language 1-9
Command Syntax 1-9
Command Prompting 1-10
Parameter Defaults 1-11

CHAPTER 2. OBJECT MANAGEMENT FACILITIES . . 2-1
Object Management Concepts 2-1
Objects 2-1
Libraries 2-2
Finding Objects through Libraries 2-5
Object Management Operations 2-7
General Object Operations 2-7
Library Operations 2-8
Damaged Objects 2-9

CHAPTER 3. WORK MANAGEMENT FACILITIES . . . 3-1
Work Management Concepts 3-2
Subsystems 3-4
Jobs 3-10
Subsystem/Job Relationships 3-14
Work Management Functions 3-16
CPF-Provided Subsystems 3-16
User-Defined Subsystems 3-18
Managing Subsystems 3-18
Managing Jobs 3-19
Initiating Jobs 3-21

CHAPTER 4. DATA MANAGEMENT FACILITIES . . . 4-1
Data Management Concepts 4-1
Files 4-1
File Description 4-2
Connecting a File to a Program 4-6
File Overrides 4-6
File Processing 4-6
Data Base Data Management 4-7
Access Paths 4-8
Members 4-11
Physical Files 4-11
Logical Files 4-12
Using Data Base Files 4-18
Device Support Data Management 4-19
Display Device Support 4-20
Nondisplay Device Support 4-27
Data Operations 4-29
Program-Described Data Files 4-29
Externally Described Data Files 4-30
Spooled File Processing 4-32
Copying Files 4-36
File Reference Function 4-37

CHAPTER 5. APPLICATION DEVELOPMENT 5-1
Overview 5-1
Design Considerations 5-1
Programming Considerations 5-5
Testing and Debugging 5-6
Documentation 5-7
Control Language Programs 5-8
Message Handling 5-11
Message Descriptions 5-12
Message Queues 5-12
Using Messages and Message Queues 5-13
Debugging Functions 5-14
Command Definition 5-15

CHAPTER 6. SYSTEM MANAGEMENT 6-1
Security 6-1
User Identification 6-2
Security Functions 6-6
Object Authorization 6-7
Using Security 6-8
Save/Restore 6-9
Save Functions 6-10
Restore Functions 6-10
Using Save/Restore 6-10
Installation and Specialization Facilities 6-11
System Operation 6-12
System Operation Functions 6-12
Message Handling 6-13
Service 6-14

GLOSSARY G-1

INDEX X-1

Chapter 1. Introduction

The Control Program Facility (CPF) is the licensed system support program for the IBM System/38. CPF is designed to complement and to 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 at 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 without disrupting applications that are already used on the system.

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 named items (such as programs or files) that are 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 does 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 subsystems that support interactive, batch, and spooling processing. Although the work management facilities can provide specialized operating environments through the use of subsystems, 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 both local and remote devices attached to the system, including many unique functions to support the display devices.

Data can be 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 once in the file rather than in each program using the file. These data descriptions are created with the use of the *data description specifications (DDS)*. A specification form (similar to the 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.

Data can also be described in the traditional manner in which 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 can be used as input by a program being tested, but are protected from being changed by the program
- 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 operator 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.

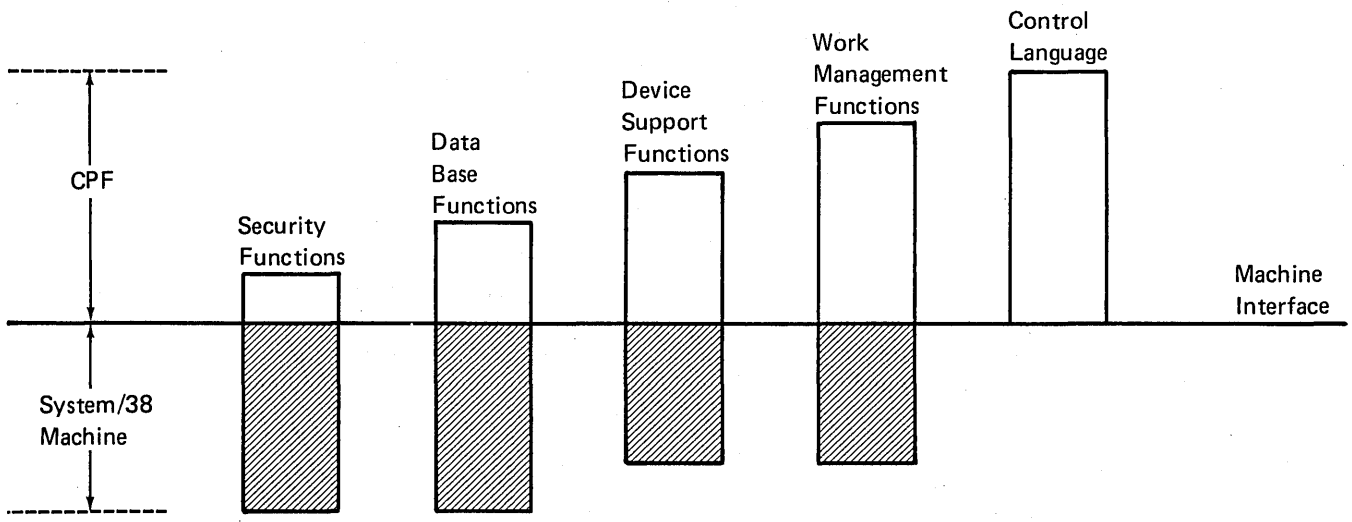
The security facilities allow individual work station users to have various levels of control over the access to objects. As security requirements change, the control provided by the security facilities can be modified.

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.

SYSTEM CONCEPTS OVERVIEW

The System/38 Control Program Facility (CPF) and other program products provide the interface between the system user and the System/38 machine. These program products capitalize on the advanced features of the System/38 machine, which includes both hardware and microcode. Many functions that have traditionally been performed by system control programs have been integrated into the machine so that they can be performed in a more basic and efficient manner. The interface between these functions and the system user is provided by the program products that use the functions.

Many CPF functions are provided through a combination of CPF and machine functions. The following drawing shows some examples of the distribution of functions between the machine and CPF. Of course, there are many other functions that are not represented in this drawing. This drawing illustrates how the degree of machine dependency varies among different CPF functions. For example, most security functions are provided by the machine, but the control language functions (which analyze and interpret control language commands) are completely provided by CPF.



Regardless of the function the system user uses, he does not need to be concerned about where the function is performed because CPF provides a single, consistent interface to all the system functions. The following sections describe some of the more significant and unique characteristics of the machine that CPF uses in providing its support.

Object-Oriented Architecture

The object-oriented architecture of the System/38 machine is fundamental to the overall design of the functions provided by the system. As described earlier in this chapter, an object is a named entity stored in the system. Included with each object is a set of attributes that describe the object. The objects that are created and used through CPF functions are built from one or a combination of the basic machine object types. For each type of object, there is a specific set of operations that can be performed on the object. Because this object-oriented architecture is a fundamental characteristic of the machine, access to an object and the operations that can be performed on it are controlled by the machine. This control provides a high degree of integrity and security not available on many previous systems.

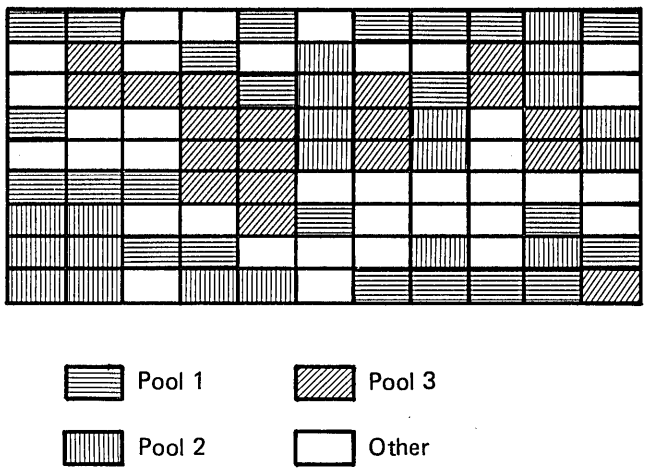
Single-Level Storage

System/38 is a virtual storage system in which all portions of main and auxiliary storage are addressed as though they were within a single address space (or level of storage). The system allows the user to access objects by name, rather than by exact storage locations. The machine uses the object's name to determine where the object exists in the system. Because operations cannot be performed on an object that is not in main storage, the machine moves all or a part of the object into main storage as it is needed and places it back in auxiliary storage when it is not needed. This transfer is not apparent to or controlled by the system user.

The transfer between main and auxiliary storage is performed in increments of 512 bytes. Each 512-byte increment is called a *page*. The process of transferring these increments between main and auxiliary storage is called *paging*. When a new page must be moved into main storage, the machine determines which existing page in main storage was used the least recently and replaces that page with the new page. If the replaced page was changed while it was in main storage, it is written into auxiliary storage so that the changes are not lost. If more than one page must be moved into main storage at the same time, the machine moves multiple pages in a single transfer operation. The paging functions of the System/38 machine provide an efficient use of main storage independent of the operations that are being performed by the system users through program products and application programs.

Because System/38 is also a multiprogramming system, main storage must be available for all the jobs that are executing concurrently in the system. To reduce the amount of interference among jobs that are competing for main storage and to prevent a very large job from using too much main storage, main storage can be subdivided for use by different groups of jobs. Main storage is subdivided according to *storage pools*, which are logical segments of main storage. When the machine retrieves an object from auxiliary storage for a job, the object (or the part of the object that is needed) is moved into main storage that is assigned to the storage pool in which the job is executing.

A storage pool provides a restricted quantity of main storage to the jobs that execute within that storage pool. A storage pool is *not* a contiguous partition of main storage. Rather, it includes a number of 1 K (1024 bytes) increments of main storage that are available to the jobs executing in it. These increments can be located anywhere in main storage. For example, the following drawing illustrates main storage that has been assigned to three storage pools:



The System/38 machine requires a certain amount of main storage for machine control functions that are always present in the system. The objects in this storage are not paged in and out during system operation. This storage is allocated to the machine when the system is started. The other machine functions are paged in and out and use a storage pool that is assigned to the machine itself (machine pool). The CPF defines another storage pool that automatically includes all the main storage that is not assigned to some other storage pool. A total of up to 16 different storage pools can be in use concurrently.

One of the fundamental elements of the System/38 machine that provides efficient use of main storage is the sharing of objects by individual users simultaneously using the system. When an object (such as a program or a data base file) is used concurrently by more than one system user, only one copy of that object is placed in main storage, even though the users might be executing jobs in different storage pools. Any number of users can be using the object. The machine provides functions that allow synchronization between users as necessary. This object sharing reduces the amount of paging performed by the system and also reduces the need for large storage pools when users are sharing an object.

Most of the storage management functions are performed and controlled by the System/38 machine. CPF provides the functions necessary for a programmer to establish the storage pools and assign jobs to them to ensure that jobs execute efficiently. These functions are provided through the CPF work management facilities. Another machine function that is controlled through the work management facilities is the activity level control.

Activity Level Control

To ensure that the processing unit is used efficiently, the System/38 machine controls the number of different jobs that are competing for use of the processing unit. The maximum number of jobs that can compete for the processing unit at one time is called the *activity level.* Jobs that are not actively competing for the processing unit (such as those that are waiting for a response from a work station) are excluded from this number.

Each job on the system is assigned to an activity level group. The System/38 machine allows up to 16 different activity level groups to be used concurrently, and allows a maximum of 16 active storage pools on the system. CPF associates each activity level group with a storage pool. The activity level group controls the number of jobs within a particular group that are competing for the processing unit.

Access Groups

Another element used by the System/38 machine to ensure that main storage is used efficiently is the *access group.* An access group is a group of system objects that provides such things as working storage areas that are needed for a job to execute. Several of these objects are needed, and system performance would deteriorate if the objects were paged in and out individually. Consequently, objects in an access group are paged in together when a job is competing for the processing unit. If the job enters a long wait (such as waiting for a response from a work station), the access group is paged out so that the main storage can be used by other jobs.

Access groups are not controlled by the system user. This activity is performed automatically by the machine to ensure efficient system performance.

Data Base Functions

Many of the data base functions that are supported by CPF are provided directly through functions of the System/38 machine. These functions allow the machine to locate records within the data base file, extract the records, and provide them to the system user. While a data base record is being updated, the machine protects the data from being changed by another system user.

In addition to being able to extract specific records, the machine can also extract individual fields from data base records and provide these fields to a system user in a format that is different from the stored format. This capability lets a system user process records in a format and sequence that may be different from the format and sequence of the data stored in the system. Along with the ability to share objects between system users, these data base functions allow interactive system users to access data in a manner that suits their own needs without affecting other system users.

The CPF data management facilities provide the interface between the system user and the machine data base functions. This interface lets the user describe data base files (with their records and fields) as well as the interface for reading, writing, and updating records in the data base.

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:

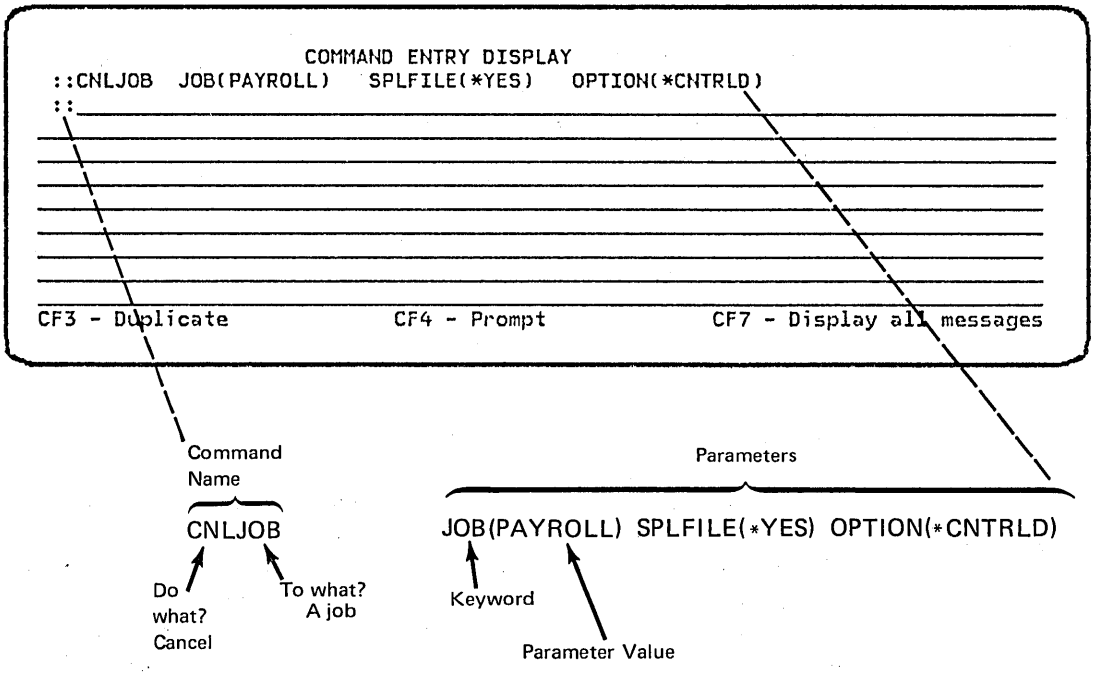
- Entered individually from a work station
- Entered as part of batch jobs
- Used as source statements to create a control language program

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 in 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 job that exists in the system.

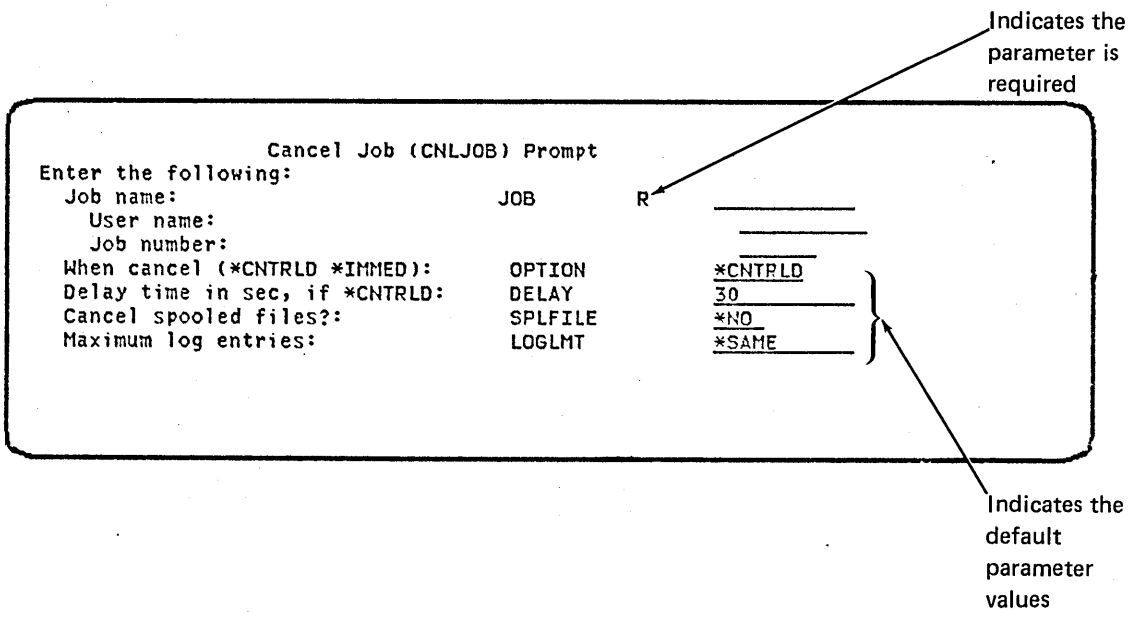
The parameters in control language commands are keyword parameters. The keyword identifies the purpose of each parameter. However, when commands are entered, the keywords are optional and can be omitted to reduce the amount of keying required. When the keywords are omitted, the parameters are positional 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 supplied with 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 in entering 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 the system supplies for the parameters that are not entered. The prompt display for the Cancel Job command shown earlier includes default values for the SPLFILE, OPTION, LOGLMT, and DELAY parameters. These default values can be changed by entering other values in their places.

Chapter 2. Object Management Facilities

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 element that is made up of a set of attributes (that describe the object) and a value. The attributes of an object include its name, type, size, the date it was created, and a text description provided by the person who created the object. The value of an object is the collection of information that is stored in the object. The value of a program, for example, is the executable code that makes up the program. The value of a file is the collection of records that makes up the file. The concept of an object simply provides a term that can be used to refer to a number of different items that can be stored in the system regardless of what the items are.

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
- Device 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 the 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. Several objects can have the same name as long as their object types differ, or as long as they exist in different libraries.

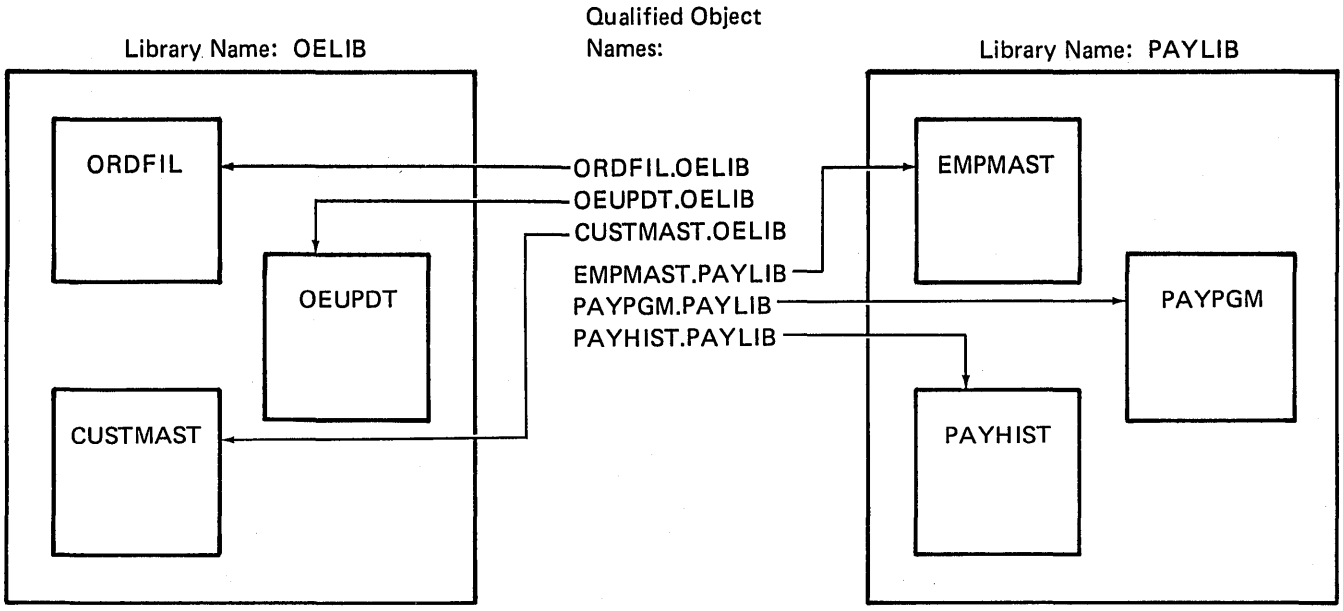
Libraries

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.

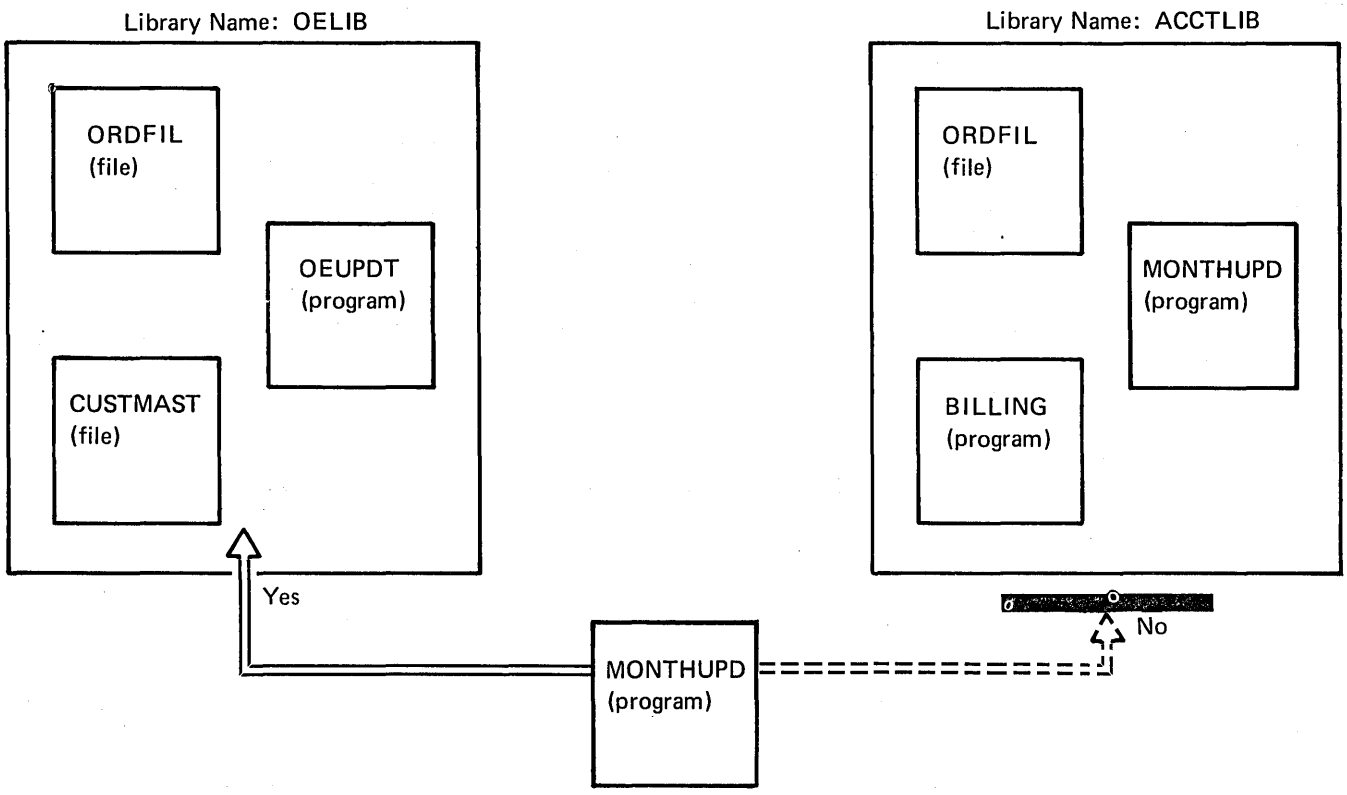
The object grouping performed by libraries is a logical grouping and does not affect an object's placement in storage. Thus, objects in a library are not necessarily adjacent to each other. The size of a library, or of any other object, is not restricted by the amount of adjacent space available in storage. 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. An object can be moved from one library to another, but a single object 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.

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 its 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. The following drawing shows two libraries and the qualified names of objects in them.



Two different objects with the same name can exist in the same library, only if their object types differ. However, two objects with the same name and type 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. For example, in the following drawing, a new program named MONTHUPD (Monthly Update) could be added to the library OELIB, but not to the library ACCTLIB.



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.

There are CPF-provided libraries and user-defined libraries. The CPF-provided libraries are:

- *The system library*, called QSYS, containing the objects that are provided as part of CPF.
- *The general purpose library*, called QGPL, 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*, called QTEMP, 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.

User-defined libraries can help the user organize work on the system for specific applications.

Finding Objects Through 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 specified, CPF attempts to find the object in the specified library. If a simple object name is specified, CPF searches a list of libraries until it finds the first occurrence of the object of that type and name or until it has searched all the libraries on the list without finding the object. The libraries that are searched, and the order in which they are searched, is 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 is called the system part of the library list. This part specifies the libraries used for all the jobs that run on the system. When the system is installed, the system part of the library list consists of only the system library (QSYS). Libraries in the system part of the library list are searched before libraries in the second part of the library list. Because the system part of the library list applies to all jobs on the system, it cannot be changed for an individual job.

The second part is called the user part of the library list. The user 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 (QGPL) and the job's temporary library (QTEMP). When a system has a number of user-defined libraries, the user part of the library list may 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 specified 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.

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 a description of an object or a group of objects. 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.

- Allocate and deallocate an object or group of objects. These operations allow a user to limit the use of an object by other users, reserve objects for use by the requester, and to free them after use. These operations are not valid for all object types.

- Change the ownership of an object. This operation allows the ownership of an object to be transferred to another user.

- Dump an object. This operation is used as an aid in debugging programs. It allows the user to dump the contents of any object stored in a library. The user must have authority for both the object and the library.

- 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 simple name of an existing object. The object itself does not change, nor does it change libraries.

- Grant, revoke, and display authority to use an object. These operations provide functions that control user access to an object and protect the object owner's rights.

- Save and restore an object, a group of objects, or an entire library. 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.

- Create or delete an object. These operations provide the functions necessary to create and delete user-defined objects.

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

Damaged Objects

If for some reason the system can no longer process an object correctly, that object is considered *damaged*. Object damage is the general term used to refer to a class of infrequent failures involving objects. These failures occur for a variety of reasons relating to internal failures in the system. Such failures include:

- Logic failures internal to the system, causing a portion of the object to be updated incorrectly.
- Hardware failures causing some portion of an object not to be read from auxiliary storage.
- System failures resulting from external environmental influences, such as power failures.

If the system is fully operable when a damaged object is encountered, the system informs the user by sending a message to the program encountering the damage. If damage to a particular object is encountered for the first time, a message is sent to the system operator.

Object damage encountered while the system is being started is communicated to the system operator by the lights on the system console, if the damage is such that the system cannot be started. These lights provide the system operator with the information necessary to restart the system.

When IBM-supplied CPF objects are damaged, some are automatically deleted and recreated by CPF, while others are deleted and restored during the installation of CPF. This ensures that, in the case of damage to an object necessary for the operation of CPF, the system is still operable.

CPF does not automatically replace user-defined objects if they are damaged. When CPF encounters a damaged object, it sends a message providing the necessary recovery procedures to the user. The user can delete these objects, using the delete object commands, and restore them from a backup copy. This allows the user to return his objects to a known, processable state. Fragments of objects and damaged objects that are no longer usable can be deleted by reclaiming auxiliary storage.

Chapter 3. Work Management Facilities

The work management facilities of System/38 CPF provide a framework through which work flows on the system and the allocation of system resources (such as main storage and processor time) are controlled. This single framework handles all types of work performed by the system regardless of how the work is submitted. To simplify the operation of the system, the specifications needed for these controls are stored in the system in CPF objects so that they do not need to be specified each time 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, including:

- Traditional batch processing. Data is collected and submitted for processing in programs that do not require interaction with a work station user. These programs are scheduled when job entry and subsystem maximum active jobs are at a level at which the additional jobs can be scheduled.

- Interactive processing of various types. This provides a wide range of possibilities, including those in which:
  - The work station user simply interacts with the system, using system-provided functions, or the user calls an application program and interacts with it.
  - The system automatically calls the appropriate application program when the work station user signs on, and the user interacts with that program.
  - The system performs different functions, depending upon the data that is provided by the work station user.

All the work performed on the system is submitted through the work management facilities. Specialized operating environments can be designed to handle different types of work to satisfy the unique requirements of an installation. These operating environments can easily be started and terminated as needed to support the work being done and to maintain the necessary performance characteristics.

This chapter describes the objects used by work management and the functions through which work is controlled on the system. When CPF is installed, it includes support for interactive, batch, and spooling processing and can be used without modification. CPF also provides the facilities through which this support can be tailored to handle unique processing requirements. Because the work management facilities can be used as they are installed, a complete understanding of work management concepts is not needed to perform typical batch and interactive operations. However, an understanding of these concepts is needed to define the environments for more advanced or specialized data processing applications.

WORK MANAGEMENT CONCEPTS

The submission, initiation, execution, and termination of work is controlled through the work management facilities. The ability to control the concurrent execution of different units of work in different environments is provided through a hierarchy of operational elements in the system. These elements are:

1. The system.

2. *Subsystems*, which are individual, predefined operating environments through which CPF coordinates work flow and resource usage. More than one subsystem can be operating at the same time in the system.

3. *Jobs*, which are the basic units by which work is identified on the system. Jobs execute within the operating environment provided by a subsystem. Each job is a single, identifiable sequence of processing actions that represent a single use of the system.

4. *Routing steps*, which are subdivisions of a job for which the execution environment can be changed. A routing step is a single processing action. (Most jobs consist of only one routing step.) The actual processing of a user's work is performed in the routing steps.

Operations performed on the system can be controlled at each level of the hierarchy. For example, a subsystem can be individually started and terminated, as can jobs within a subsystem. When a subsystem is terminated, any jobs executing in it are terminated.

Each of the operational elements, except the system itself, is defined through an object. Although no object exists through which the system as a whole is defined, certain specifications (such as the system date, system date format, and system time) are defined for the entire system. These specifications are contained in *system values*. Figure 3-1 shows the relationships between the operational elements and the objects that define them.

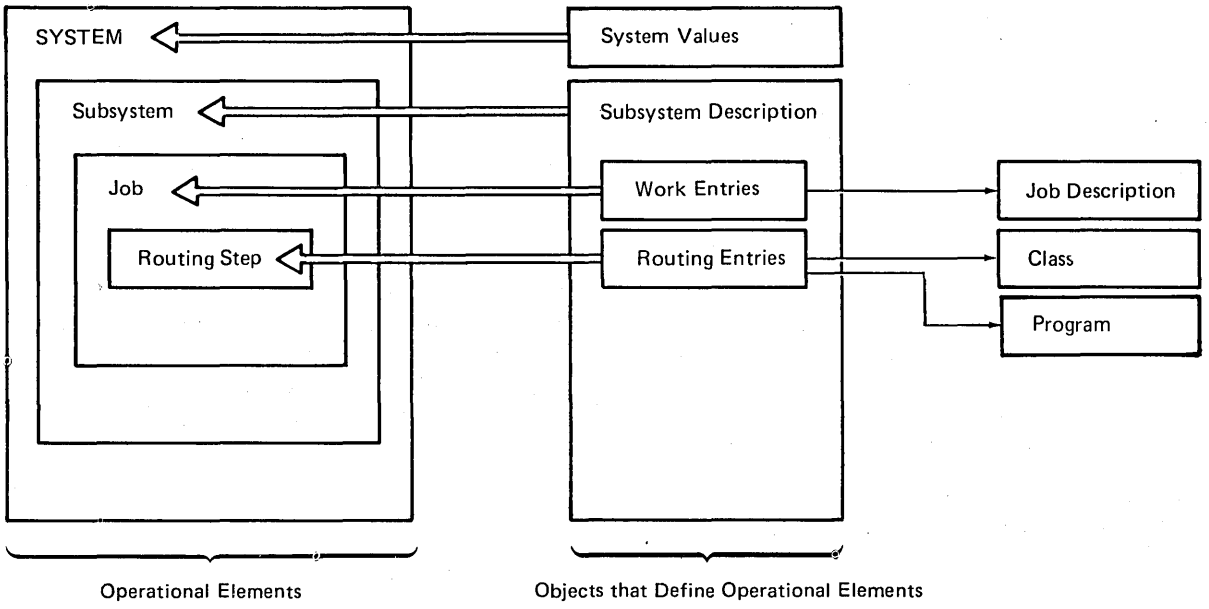


Figure 3-1. Operational Elements and the Objects That Define Them

Once the system is started, the system operator can start and terminate individual operating environments by starting and terminating subsystems. Work is then performed by jobs that execute within the subsystems. Jobs are considered to be *batch* or *interactive*, according to the way they are initiated. Interactive jobs are initiated when the user signs on at a work station. Batch jobs are initiated by the use of job control commands.

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 requests to be performed. 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*; they are submitted to the system and then are selected from the queue by the work management facilities. Although a batch job might access a work station for all or part of its input data, it is not treated as an interactive job by the system because the job was not initiated by a sign-on at a work station.

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 only through the use of job steps, which are identified in the control statements that make up the job. However, in System/38, programs can simply call other programs directly. Thus the job is simply made up of whatever 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

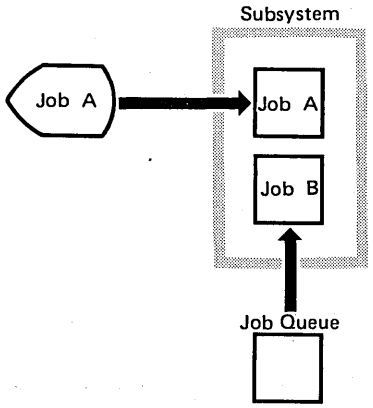
All jobs processed in the system execute within an operating environment called a *subsystem*. The specifications that define the subsystem, and that CPF uses to control the subsystem, are contained in an object called a *subsystem description*.

Subsystems

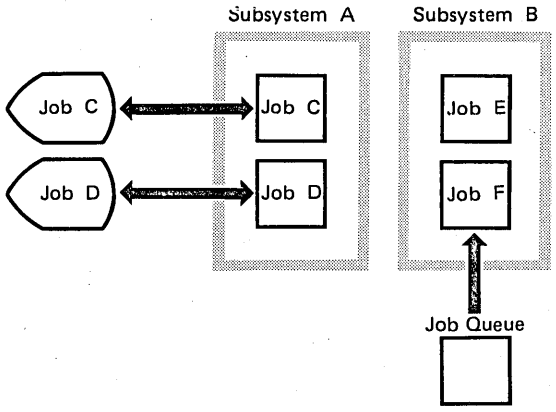
The environment of a subsystem includes main storage and activity level controls, sources from which jobs can be accepted, and programs that can be invoked by the subsystem to start the execution of the job. These programs can, in turn, call other programs to perform the functions required. Although 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.

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 (initiated) at one time is limited only by the resources available on the system. Although any number of subsystems can be active concurrently, at least one active subsystem is required at all times. Each subsystem can be started and controlled independently. One subsystem, called the *controlling* subsystem, is automatically started when CPF is started.

Both batch and interactive jobs can execute in a single subsystem, or separate subsystems can be used for each type of job. The following drawing shows both types of jobs executing in one subsystem. Job A, an interactive job, is being used by a work station user. Job B, a batch job, was selected from a job queue and is operating concurrently with Job A in the same subsystem.

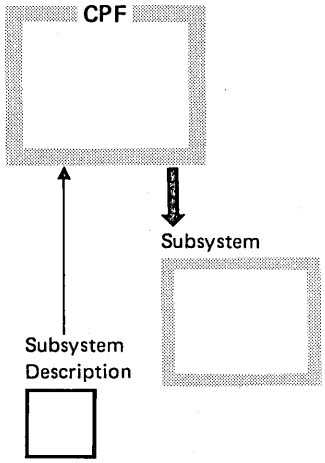


In the following drawing, two subsystems are active concurrently. Interactive jobs C and D are executing in subsystem A. Two batch jobs, selected from a job queue, are executing in subsystem B. The use of separate subsystems, as shown here, allows each operating environment to be individually controlled.



Subsystem Descriptions

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 for 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 upon several factors, such as:

- The number of unique processing environments that are needed to support various applications
- The level of operational control that is needed over the applications that operate within the different subsystems
- The amount of isolation that is needed between the 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. This implementation of subsystems provides the following advantages:

- The processing environment needed for 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 performance predictability can be achieved within a subsystem because the use of resources by that subsystem can be isolated from the use of resources by other subsystems. This ensures that applications with specific performance requirements, such as interactive applications, have the resources needed to operate in a uniform manner.

A subsystem description contains the following three categories of information:

- Subsystem attributes, which specify the 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 and attributes for the jobs selected from those sources
- Routing entries, which specify the programs that can be invoked by the subsystem and the execution environment in which those programs are to operate

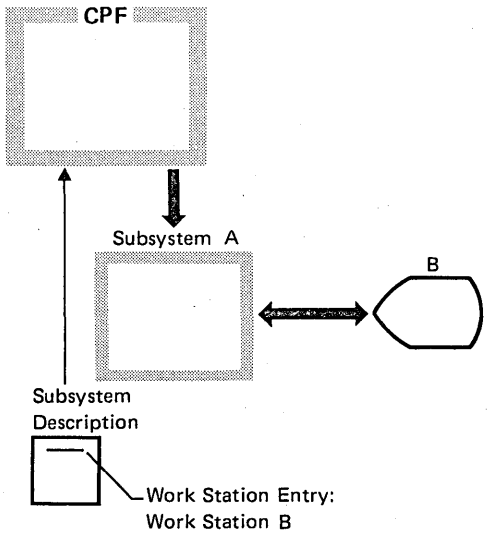
Subsystem Attributes

The subsystem attributes specify the storage pools available to jobs executing in the subsystem and the maximum number of jobs that can be active (initiated) concurrently in the subsystem. Each subsystem can have its own unique storage pools or a subsystem can share a common (base) storage pool with other subsystems. Many CPF programs execute in the base storage pool, even though the routing steps using these programs are executing in other storage pools. In addition to the subsystem activity level, each storage pool has an activity level associated with it, which limits the number of routing steps in the storage pool that can be competing for the processing unit.

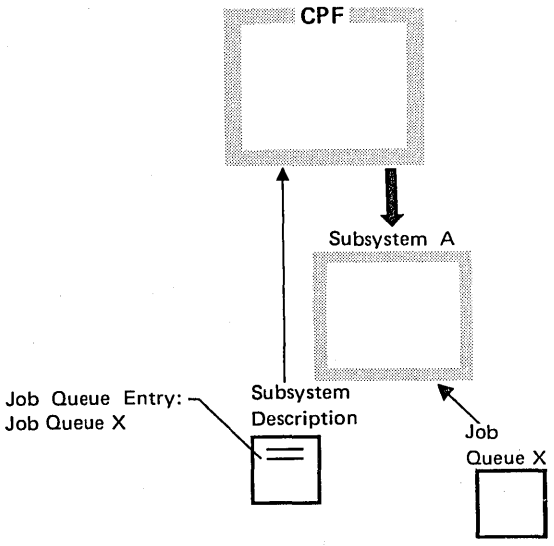
Work Entries

The *work entries* in a subsystem description specify the sources from which jobs can be selected to execute in the subsystem. The work entries, for interactive and autostart jobs, also specify a job description that provides default attributes for the jobs initiated from each work entry.

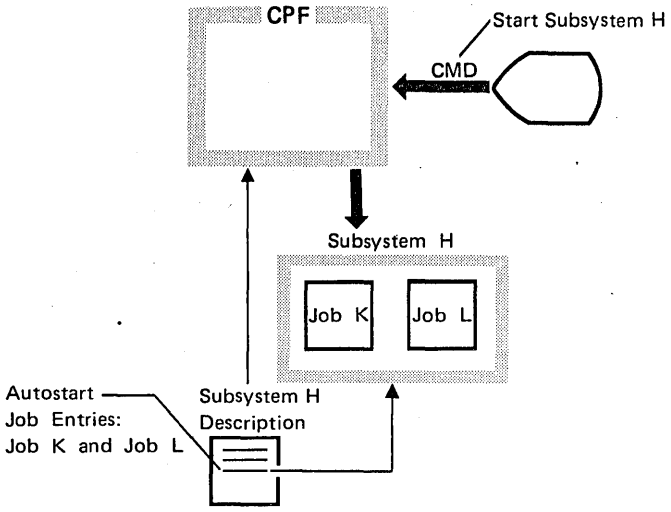
Work Station Entries: Work station entries specify the work stations from which interactive jobs can be initiated in the subsystem. An interactive job is initiated in the subsystem when a user signs on at one of the work stations specified in the work station entries of the subsystem description. For example, in the following drawing, work station B is specified as a work entry for subsystem A.



Job Queue Entry: The job queue entry specifies the job queue from which the subsystem can initiate batch jobs. Only one job queue entry is allowed in a subsystem description. Jobs are placed on the job queue when they are read by a spooling reader or submitted to the queue from another job. In the following example, the job queue entry specifies job queue X as the job queue for subsystem A.

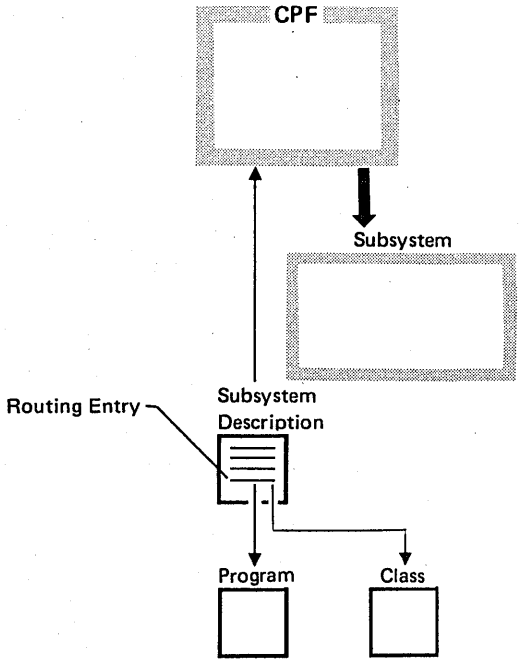


Autostart Job Entries: Autostart job entries specify jobs that are to be automatically initiated when the subsystem is started. The jobs specified as autostart jobs are not initiated from a work station, so they are processed as batch jobs. (However, because they are directly initiated by the subsystem, they are not placed on a job queue.) Autostart jobs can be used to perform initialization, recovery, or other functions for applications or other jobs that execute in the subsystem. In the following drawing, jobs K and L are started automatically when subsystem H is started.



Routing Entries

The routing entries in a subsystem description specify the programs to be invoked when routing steps are initiated, and specify the classes that define the execution environment in which the programs are to execute (see Figure 3-1).



The routing entries in a subsystem description form a *routing table*. When a job is initiated, the appropriate routing entry is selected by means of routing data. The routing data can be entered by a work station user after he starts an interactive job, extracted from the job description associated with the job, or specified when a batch job is placed on a job queue. CPF compares the routing data with compare values in the routing entries in the routing table to determine which routing entry is to be selected.

The processing performed as a result of invoking the program specified in a routing entry is called a *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 are performed in the routing step, or the program might pass control of the routing step to another program.

For example, the subsystem descriptions provided by CPF specify the CPF control language processor (program QCL) in the routing entries. Thus the control language processor is invoked when routing steps are initiated in these subsystems, and work can be submitted through control language commands. However, to make the system easier to use, the control language processor often invokes other programs, depending upon the user who signed on. When the system is installed, the following programs are invoked by the control language processor:

- The program that displays the command entry display is invoked if the security officer signs on.
- The program that displays the programmer's menu (on the 5251 work station, Model 11 or 12) or the command entry display (on all other work stations) is invoked if the programmer signs on.
- The program that displays the system operator's menu is invoked if the system operator signs on.
- The program that displays the program call menu is invoked if authorized work station users sign on.

Jobs

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

- Interactive jobs are initiated when a work station user signs on. Routing data used to initiate the routing step in the job can be either entered by the work station user or extracted from the job description that is specified in the work station entry of the subsystem description.
- A batch job is initiated when the subsystem selects a job from the job queue. The job could have been read by a spooling reader and placed on the job queue, or it could have been submitted to the job queue from another job. In either case, the routing data is specified for the job or extracted from the job description. Batch jobs, once initiated, are usually controlled by the system operator.
- Autostart jobs are initiated automatically when a subsystem is started. The routing data used to initiate the routing step in the job is extracted from the specified job description.

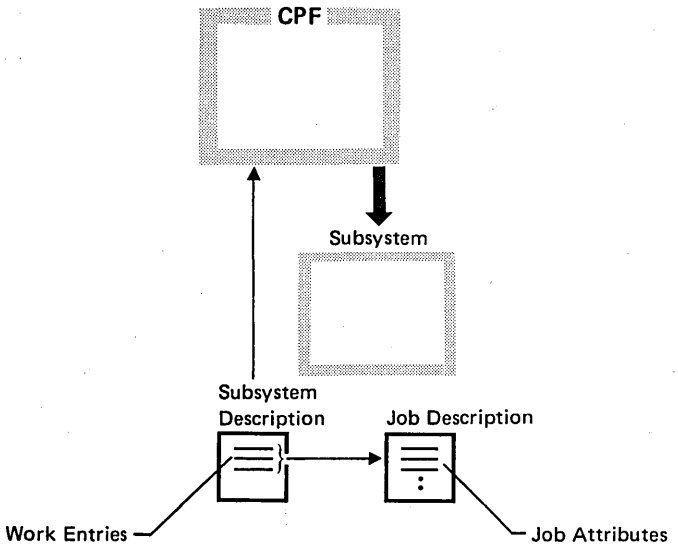
Although interactive and batch jobs appear to be quite different to the system user, all jobs once started, are handled essentially the same way by the system. Each job exists and is identifiable in the system from the time it is submitted (for example, from sign-on at a work station or from the time 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 can be used 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 are needed for different jobs to meet the special requirements of each job. 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 predefined. These attributes can be modified as processing needs change. The attributes specified in a job description include:

- The job queue on which the job should be placed when it is submitted (for batch jobs only).
- The scheduling priority used to control when the job is selected from the job queue for execution.
- The user portion of the library list that is in effect when the job is started. (There are two parts of a library list: the user portion and the system portion. Only the user portion of the library list is changed.)
- 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.
- The output scheduling priority to be used for producing spooled output.
- The user profile for the job.

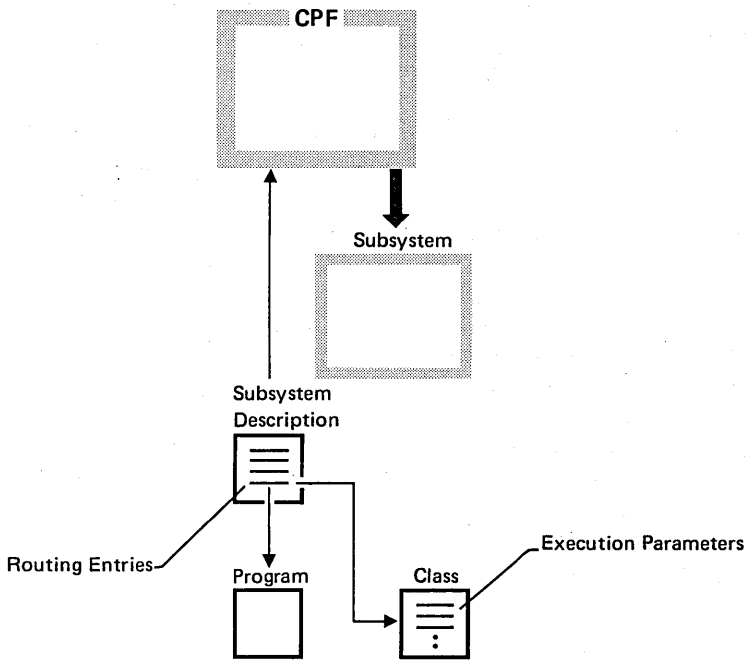
Each work entry in a subsystem description except the job queue entry refers to a job description for job attributes (see Figure 3-1). CPF is shipped with pre-built job descriptions as default values for batch, spooled, and interactive jobs.



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

Routing Steps

Work management establishes the execution environment for a routing step when the routing step is initiated. The execution environment of a job can be changed during a job because that environment is defined by the routing entry, not by the job description. (The job description only defines the attributes, or external characteristics, of the job.) The execution environment is specified through an object called a *class*, which is specified in the routing entry (see Figure 3-1).



The same class can be specified for any number of routing entries. The parameters that can be 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

Most jobs consist of just one routing step, the one initiated at the start of the job. If a routing step's execution environment is to be changed, a new routing step is initiated. This may change the program that controls the routing step, the storage pool in which the program executes, the class that describes the execution environment, or the subsystem in which the routing step executes. In any case, only one routing step is ever active at a time in any job.

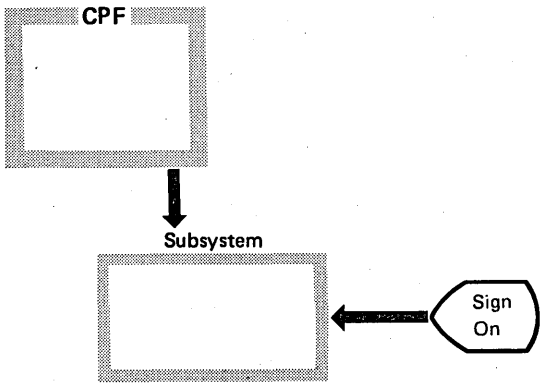
Subsystem/Job Relationships

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

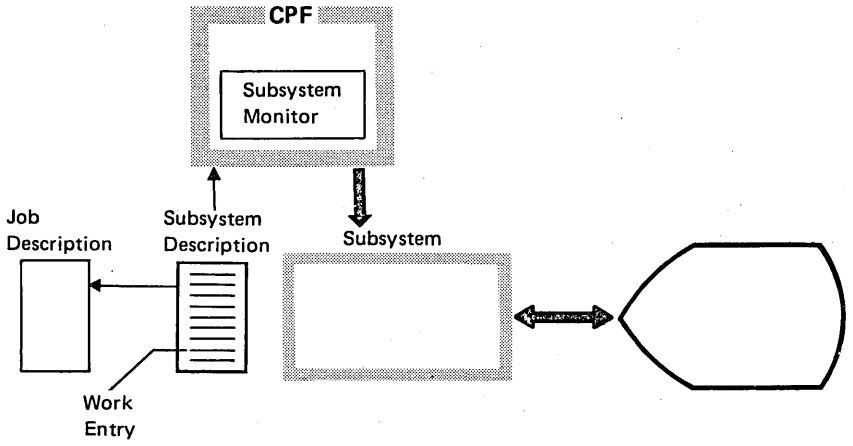
Because operating environments (subsystems) are defined through these objects, a single subsystem monitor is able to manage all types of environments. A separate invocation of this monitor is established by CPF for each subsystem that is started. Each invocation uses a separate subsystem description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

The work management facilities control the execution of jobs within the subsystems that are active on the system. The following example shows how the work management facilities use information in the various objects to start an interactive job. This example shows how a job would be started using the support for interactive processing provided when the CPF is installed. When this example begins, CPF and the subsystem have already been started. The following steps take place:

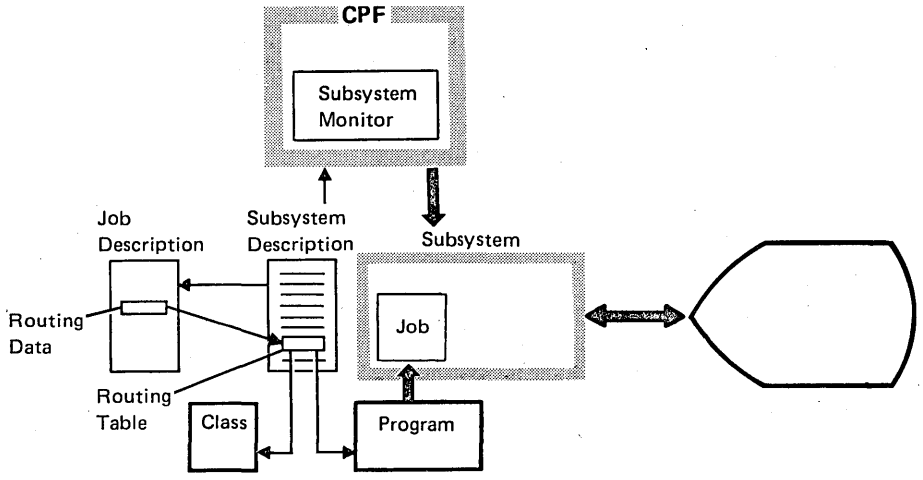
1. The work station user signs on the system.



2. The subsystem monitor uses the work station entry in the subsystem description to determine which job description to use.



3. Using the routing data from the job description, the subsystem monitor finds the appropriate routing entry in the routing table. This routing entry specifies that the program to be invoked to control a job should be invoked for the routing step, and also identifies the class that specifies the execution environment for the routing step.



4. The user can now enter control language commands to perform his work. For example, he might use a command to call an application program. When that program ends, he can enter other commands to perform other operations.

This sequence shows the functions performed by work management. These functions are performed automatically when the work station user signs on. No action by the system operator is required, and other users of the system continue to operate independently of the new interactive job.

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 subsystem descriptions designed to provide interactive, batch, and spooling operating environments. These subsystems can be used as installed to satisfy typical processing requirements.

Interactive Subsystem

The interactive subsystem, QINTER, provides an operating environment that supports jobs processed interactively through the system console or through work stations. Another subsystem, QPGMR, is available to programmers for online programming. When CPF is started, a *controlling subsystem* is automatically started. When the system is shipped, QCTL is specified as the controlling subsystem. QCTL supports all interactive jobs processed through the system console.

Batch Subsystem

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

Spooling Subsystem

The spooling subsystem, QSPL, 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 when the system is terminated 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, display, and delete subsystem descriptions and to add, remove, and change specific entries in them. These functions can be used either to change the CPF-provided subsystem descriptions or to create other subsystem descriptions to support 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. For example, if certain work stations are to be used only during a specific period of the day, these work stations could be specified as work entries in a user-defined subsystem that the system operator starts and terminates on a set schedule each day.

- Provide more than one active job queue on the system. Because a subsystem can select jobs from only one job queue, more subsystem descriptions (each specifying a different job queue) could be created. The subsystems could then be started and terminated to process different kinds of jobs, such as:
  - Long-running batch jobs
  - Nighttime batch jobs
  - Very-high-priority batch jobs
  - Batch jobs requested by the system operator

- Provide specific control over a critical or unique application. By having a separate subsystem for this type of application, the performance and consistency of the application can be controlled more easily.

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 subsystem description.

- Adding, changing, or removing work entries in existing inactive subsystem descriptions. Separate commands apply to each type: autostart job entries, work station entries, or a job queue entry.

- Adding, changing, or removing routing entries in existing inactive subsystem descriptions.

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. The job being held must exist on the system as an active job or as a job on a job queue.
- Releasing a job. This operation makes a previously held job available for further processing. The job being released must exist on the system as an active job or as a job on a job queue.
- 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.
- Signing 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.

- Returning from a routing step. This operation returns control from the most recent invocation of the program (user- or system-supplied) to the next highest program invocation. If the most recent invocation is the highest in the series of invocations (called the *invocation stack*), the following results occur:
  - The routing step is terminated.
  - Control is returned to the subsystem monitor. This will either reroute the job, thus initiating a new routing step if the job is to be automatically routed, or display the manual routing screen if the job is to be manually routed.

- 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

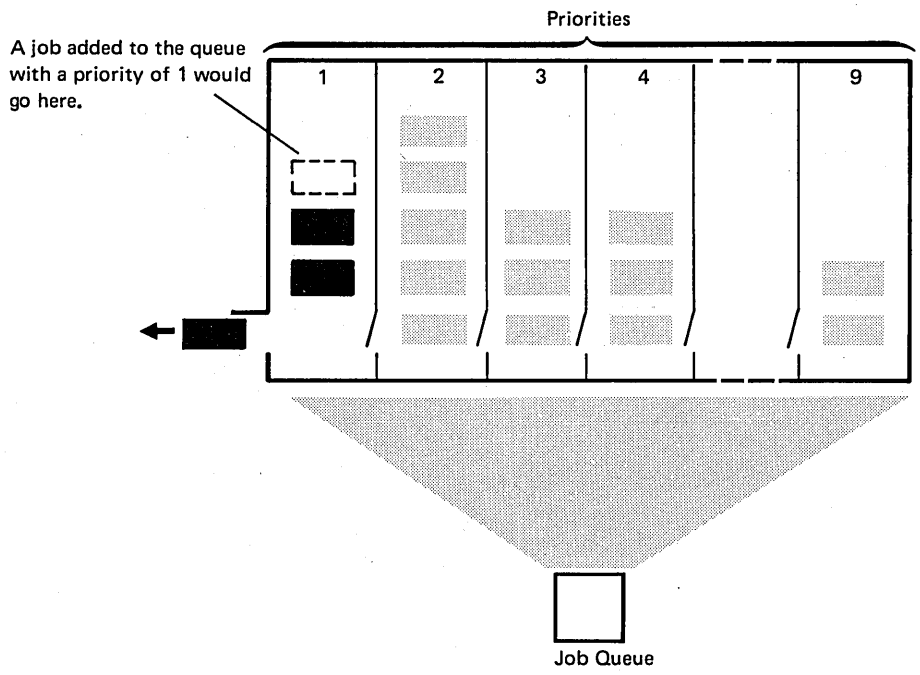
Interactive Jobs

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 allocated to an active subsystem when jobs are placed on it, the jobs remain on the queue until a subsystem that specifies the queue is started.

Jobs are selected from a job queue according to the scheduling priority assigned to each job. The scheduling priority is assigned in the job description specified at the time the job was submitted to the queue. More than one job from the job queue can be executing concurrently. The maximum number of active jobs from the job queue is controlled by the maximum number of active jobs specified in the job queue entry in the subsystem description. As shown in the following diagram, jobs with a higher scheduling 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 job queue does not have to be in use by a subsystem at the time. The function could be performed within the submitting job, but it might delay the work currently being done.

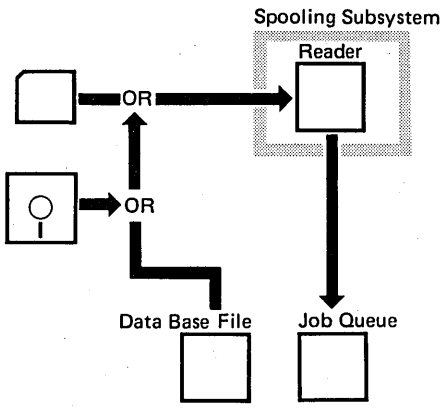
For example, if the system operator needs to save a data base file, he can simply enter the appropriate command. When the file is completely saved, he can enter other commands to continue his work. The operator may need to save the file, but may not want to wait until the end of the operation to continue his other work. In this case, he can submit the job to a job queue, to be performed independently of his other work. The job will be selected from the job queue by the appropriate active subsystem.

Submitting Jobs Through a Reader: Batch jobs can be placed on job queues 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 Chapter 4, *Data Management Facilities*.

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

- Cards
- Diskettes
- Data base files

The following drawing shows how a reader 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 example shows an input 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 JOBPTY(2)
RPLLIBL  LIBL(ORDLIB QGPL QTEMP)
CALL  SALES205                    /*PRINT DEPT SUMMARY*/
CALL  SALES206                    /*PRINT PRODUCT SUMMARY*/
CALL  SALES209                    /*PRINT SPECIAL SALES*/
// DATA
      AJ1052
      XQ4031
      BZ9504
        .
        .
        .
//                                /*END OF INLINE DATA*/
CALL  SALES213
// ENDJOB
// JOB DAILYSHIPT JOBPTY(5)
        .
        .
        .
// ENDJOB
```

If readers are used to submit batch jobs with or without inline data files, 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.

Chapter 4. Data Management Facilities

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 associated data is stored permanently in the system. *Device files* are files whose associated 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
- Tape files
- Communications 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 communications 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.

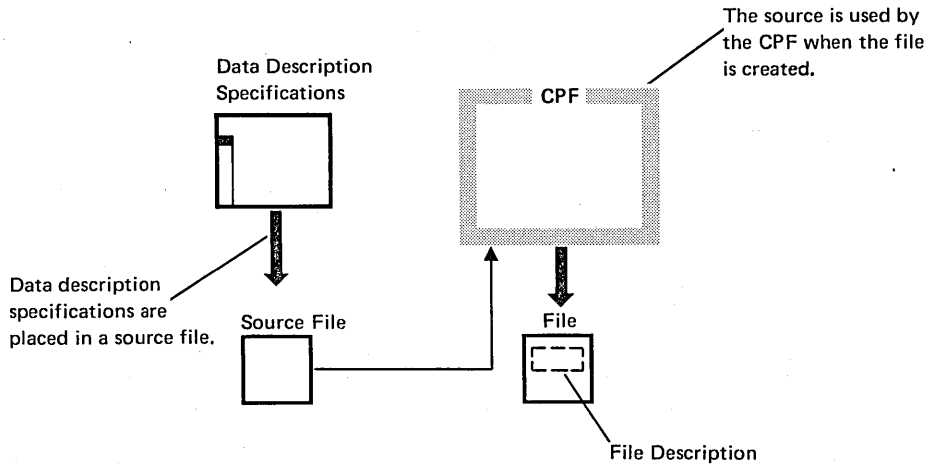
File Description

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 upon 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.

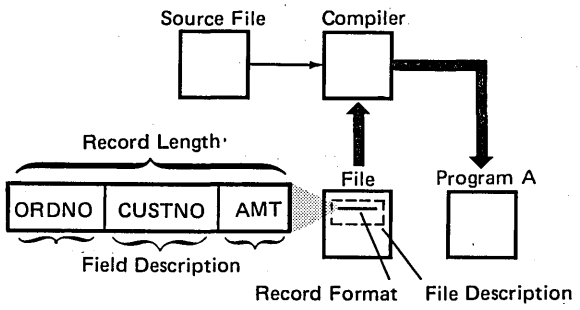
The data contained in a file is described in the file description. The record format specifications can describe the fields contained in the record. A file whose data is described at the field level in the file description is called an *externally described data file* because the data is described apart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



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 files, printer files, and communications files. Program-described data files are supported for all files except communications 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 normally 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, if that program contains its own record format specifications. 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.

Each record format is assigned a level identifier when the file it is associated with is created. This level identifier is the date and time the file was created. The level identifiers of the opened file and the file description that is part of the compiled program are compared when the file is opened. If the identifiers differ, it indicates that the file was changed (deleted and created again) and the changes could affect 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 and record formats that are not used by a program are changed, the program does not have to be changed or compiled again. In many cases, if a record format used by the program is changed, the program can be compiled again without requiring any changes to the program's source statements.

- 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. This occurs 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, supporting inquiries about the files and record formats.
  - Integrity is improved 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 with minimal fields. When a program processes the file, records are passed to and from the program. 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 in the same routing step while the file is being processed

Record Format Level Checking: Record format level checking provides a way to ensure that the record format specifications for an externally described data file 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 compiled. 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 the file type (such as data base files or card 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 provides access to data processed by the readers and writers; a spooled file is not intended for direct access to a device. When spooled device files are opened for input, they provide access to data that is read inline with a job. When they are opened for output, the data is stored by CPF as spooled files on an output queue until it is written to the device by a *writer*.

Output 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 automatically allocates the file to the program. Data base files can be allocated exclusively to a program or shared by different jobs so that more than one user can access the file at the same time. Files can also be explicitly allocated to a job (before the file is opened) as a result of a control language command. Explicitly allocating a file to a job ensures that the programs in the job will have access to the necessary 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 used to temporarily change attributes of a file that were specified when the file was created. File overrides are specified through certain commands when a file is used. The overrides are performed when the file is opened, and they remain in effect for that file until it is closed or until a job or program is completed. Overrides can be used for the following:

- Changing the name of the file to be processed
- Indicating whether input or output is to be spooled
- Redirecting input and output to different devices (for example, sending printer output to a different printer)

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 may 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 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:
  - A record can be read while being updated by another job.
  - A record can be updated by only one job at a time; this guarantees data integrity for the record.
- Storage space is used more efficiently because less redundant data 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 physical file has a fixed format and an implied access path. 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. 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, in which the records are retrieved consecutively from the file
- Directly by relative record number, in which 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

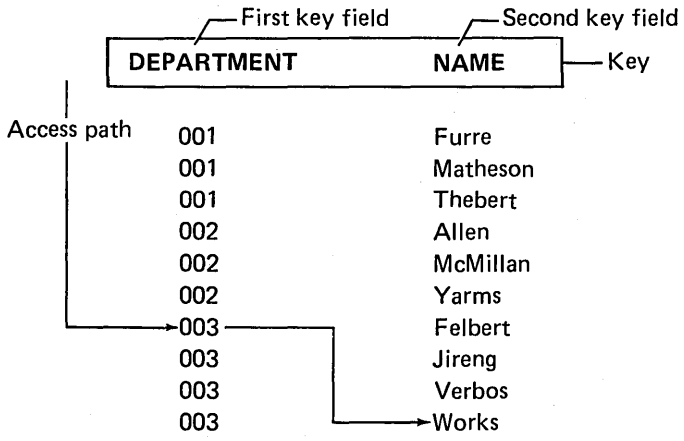
A keyed-sequence access path is created when the file is created. It is updated whenever records are added to or deleted from a file or whenever a record is modified and the contents of a key field change. Thus, all the access paths to the data can be maintained concurrently.

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 or prohibited 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 preceding or succeeding records sequentially. Thus, a work station user (through an application program) could randomly select a record, then continue with sequential processing or select records based on their position in relation to the randomly selected record.

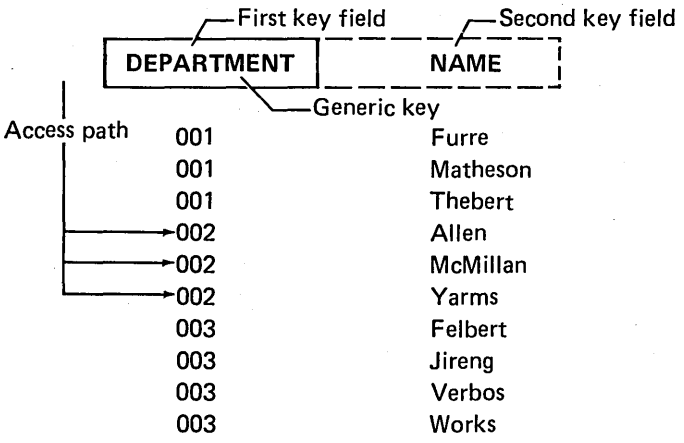
If more than one key field is used for the keyed-sequence access path, either the entire key or a partial, or *generic*, key can be used to retrieve the records. If a generic key, which has fewer fields than the entire key, is used to retrieve the record, the first record that contains the requested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

For example, a company keeps employee information listed by department number and employee name. If a manager needs to know how many days of vacation a particular employee has taken so far this year, he finds the information by first obtaining the information for his department, 003, and then for the employee, Works.



In this case, the entire key is used in accessing the records.

At another time, an employee in the Payroll Department needs to know how many hours of overtime were worked by department 002. In this case, a partial, or generic, key (department number) is used to retrieve the records.



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 added 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; however, a logical file member can be used to merge records from more than one physical file member so that 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.

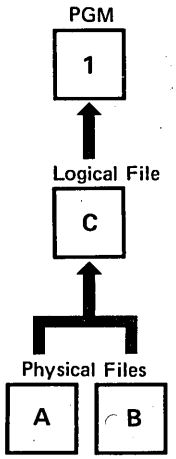
The file description for a physical file is created when the file is created. The file description includes:

- 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.

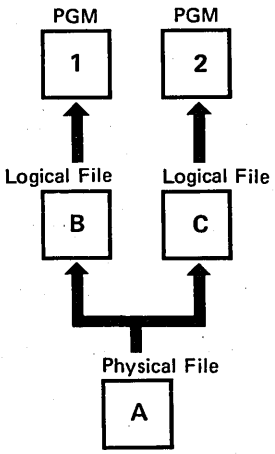
Logical Files

A *logical file* is a data base file through which data that is stored in one or more physical files can be accessed. The data can be accessed through record formats and/or access paths that are different from the physical representation of the data in the system. 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 logical 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 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 records sequenced on the same key field in the same order, they can share the same access path. When this is done, the access path needs to be described only once and 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. In DTLIN, 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.

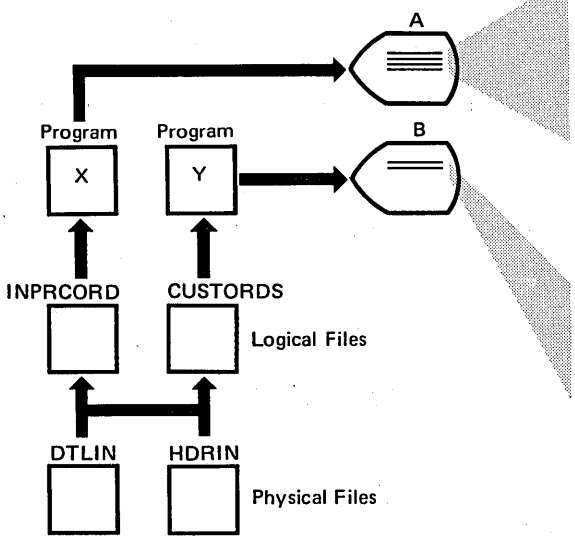
PHYSICAL FILE: HDRIN

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
	566421	071778	108211	085	067152

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

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).



ORD NO: 523480 DATEIN: 07/10/78 CUSTNO: 100429 SALESMAN: 026

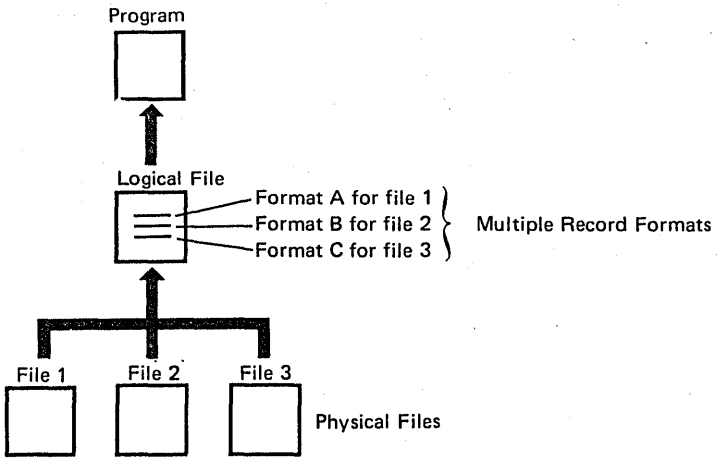
LINENO	ITEMNO	QTY	AMT	INSTK
01	31482	024	48.00	YES
02	42218	481	962.00	YES
03	22184	006	108.00	NO
04	18421	037	314.50	YES

CUSTNO: 108211 CREDIT CLASS: 02

ORDNO	DATEIN	TOTALAMT
522184	06/30/78	1411.11
566421	07/17/78	671.52

Multiple Record Formats

A logical file can use more than one record format. Each logical file record format must be related to one or more physical file record formats and can be used for processing data from one or more physical files. If a logical file record format is used to process data from more than one physical file, all the fields in the logical file record format must be contained in each of the physical file's record formats. One logical file *record* cannot contain data from more than one physical file. A single logical file with multiple record formats can be used to process data from more than one physical file 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.

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.

In the example shown previously under *Logical Files*, file transformation occurs when records for the logical file CUSTORDS are retrieved from the physical file HDRIN. The physical file description specifies an arrival-sequence access path and the record format containing the following:

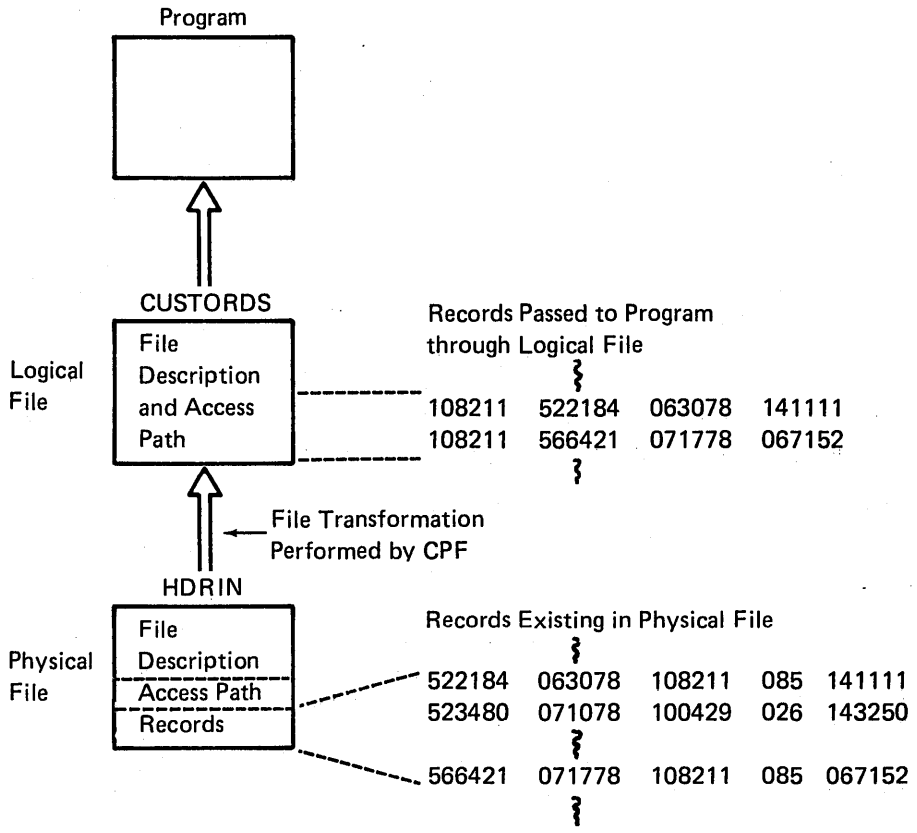
```
ORDNO ORDDAT CUSTNO SALMN TTLAMT
```

The logical file CUSTORDS does not use the field SALMN and uses the other fields in a different order. The record format specified in the file description contains:

```
CUSTNO ORDNO ORDDAT TTLAMT
```

Because this file is used to find records according to the customer number and the order date, those fields are specified as key fields in the file description and are used to build a keyed-sequence access path for the file.

When a program requests a record from the file CUSTORDS (by providing the customer number), CPF finds that record through the access path, the record format changes from the format used in the physical file HDRIN, and CPF passes the record to the program. This is done each time a record is requested by the program. The following drawing shows how this file transformation occurs:



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.

Because the same physical data in the system can be processed differently through different logical files, the copy, sort, and file maintenance operations that are often necessary on other systems can be avoided. Thus, fewer file preparation and maintenance activities are required when programs are executed, and a minimum of redundant data needs to be 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.

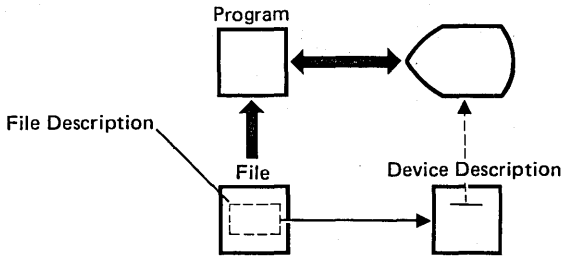
DEVICE SUPPORT DATA MANAGEMENT

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 communications devices, the diskette magazine drive, the multifunction card unit, the magnetic tape unit, and the system and work station printers.

For each device attached to the system, there is a *device description* object that describes the device to CPF.

Data management operations access external devices through *device files*. Device files contain file descriptions, which are kept in the system and are used by CPF to transfer data to and/or from the device. The same device file can be used concurrently by more than one job. Each job must use the device file with a different device.

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.



Both externally described data files and program-described data files can be created for work stations and printers. Only program-described data files can be created for card, diskette, and tape devices. For communications devices, only externally described data files can be created. The formats of records in program-described data files must be described in the programs that use the files. The record formats of externally described data files are described to CPF when the files are created and are kept in the file descriptions.

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:

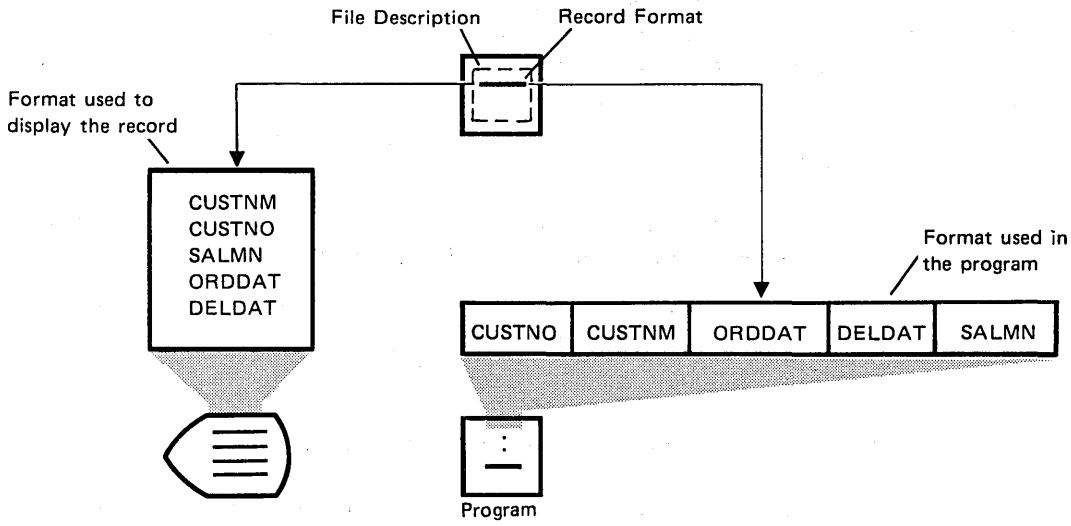
- CPF performs the device control operations, including formatting data on the screen, accepting input from the keyboard, and handling error conditions that occur at the device.

- CPF can perform subfile operations, which let the program perform input/output operations that send and receive multiple records of the same record format 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 an invalid numeric value into a field whose range 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 as it is 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.

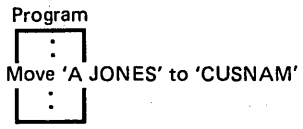
Using Display Device Support

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:

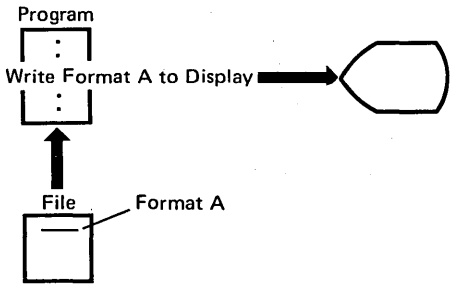
- Format displays on the screen so that the work station user can use the information provided by the program. This function includes the capability to control, from the application program, which fields are displayed on the screen.
- Design displays into which the work station user can easily enter input.
- Automatically associate the device file used by a program with the work station from which the program is invoked.
- 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.

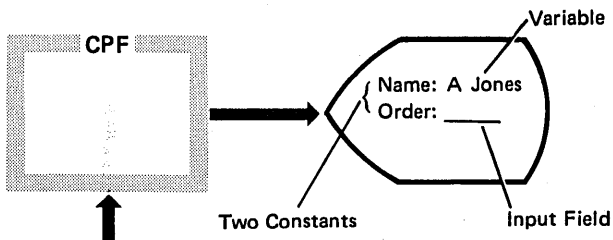
1. The application program moves the data A JONES into a variable named CUSNAM.



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 Order:) are specified in the record format. The variable from the application program is displayed following Name:. The input field following Order: is to be filled in by the work station user.



DATA DESCRIPTION SPECIFICATIONS

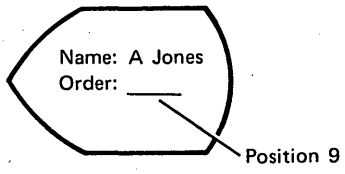
A

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Condition Name: Not (N)	Condition Name: Indicator	Condition Name: Not (N)	Condition Name: Indicator	Name Type (blank/R/K/S/O)	Reserved	Name	Length	Reference (R)	Data Type (blank/A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (blank/O/I/B/H/M)	Location: Line	Location: 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	30 31 32 33 34	29	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																1	2	'Name:'
	A										CUSNAM	20					1	8	
	A																2	2	'Order:'
	A										ORDNUM	9			0I		2	9	CHANGE(01)
	A			07															ERRMSG('Order not found' 07)

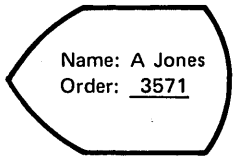
Two specified constants

Variables

4. The cursor is placed at the first position of the input field (position 9).



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.

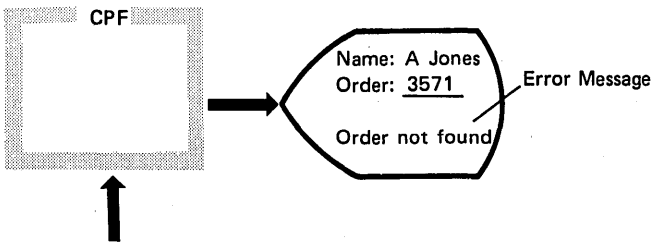


6. CPF places the data, 3571, into the variable ORDNUM and turns on indicator 01 to indicate that the field has been changed.

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

   The indicator included in the ERRMSG keyword, 07, is always turned off when the record is returned to the program. This eliminates the need for the program to reset error indications.



A

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Name Type (blank/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (blank/A/P/S/B/A/S/X/Y/N/I/W)	Decimal Positions	Usage (blank/O/I/I/B/H/M)	Location: Line	Location: Pos	Functions
	A																1	2	'Name: '
	A										CUSNAM		20				1	8	
	A																2	2	'Order: '
	A										ORDNUM		4		0	I	2	9	CHANGE(01)
	A			07															ERRMSG('Order NOT FOUND' 07)

Indicator

Error Message

Nondisplay Device Support

The nondisplay device support functions are the data management functions that apply to printers, the diskette drive, the magnetic tape unit, the communications devices, and the multifunction card unit. The file description for a nondisplay device file includes:

- Identification of the device associated with the file.
- Spooling information, such as output scheduling, copies, and forms for the file.
- Device-dependent information. Examples include form size and number of lines for printer use, and diskette location for diskette use.

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 printers attached to System/38 are supported by CPF through printer files. Among the functions supported for printer files are:

- Folding or truncating the output records if the lines of the records passed to the printer are longer than the maximum line length allowed for the device. Folded records are continued on subsequent print lines until the entire record is printed.
- Spooling the file to an output queue for subsequent printing. When the file is spooled, execution of a program that uses the file is not dependent on the availability or speed of the printer.
- 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. When externally described data printer files are used, the format of the printed records can often be changed without changing the programs that use the file. When the record format needs to be changed, the system user changes the file description by creating the file with a different record format, then recompiling the program.

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. (This function is handled by the high-level language being used.)
- Combined files can be used that combine reading input data and punching and/or printing output data. (This function is handled by the high-level language being used.)
- Output records that exceed the length allowed by the device are truncated to the maximum length allowed.
- Files can be copied from cards to the system and from the system to cards.

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, copy files from diskettes into the system and from the system to diskettes, clear diskettes, and duplicate diskettes. Multivolume diskette files are also supported. Diskette device files are always processed as program-described data files.

Magnetic Tape Unit Support

Tape is supported through high-level language programs for both input and output files. CPF includes the functions needed to initialize tape, display the volume and file label information from a tape, and copy files from tapes into the system. Multivolume tape files are also supported. Tape device files are always processed as program-described data files.

Communications Devices

Communications devices are supported through high-level language programs for combined output and input files. CPF includes the functions needed to open files for output/input operations, obtain records from output/input files, write records to output/input files, and close files. Communications files are always processed as externally described data files.

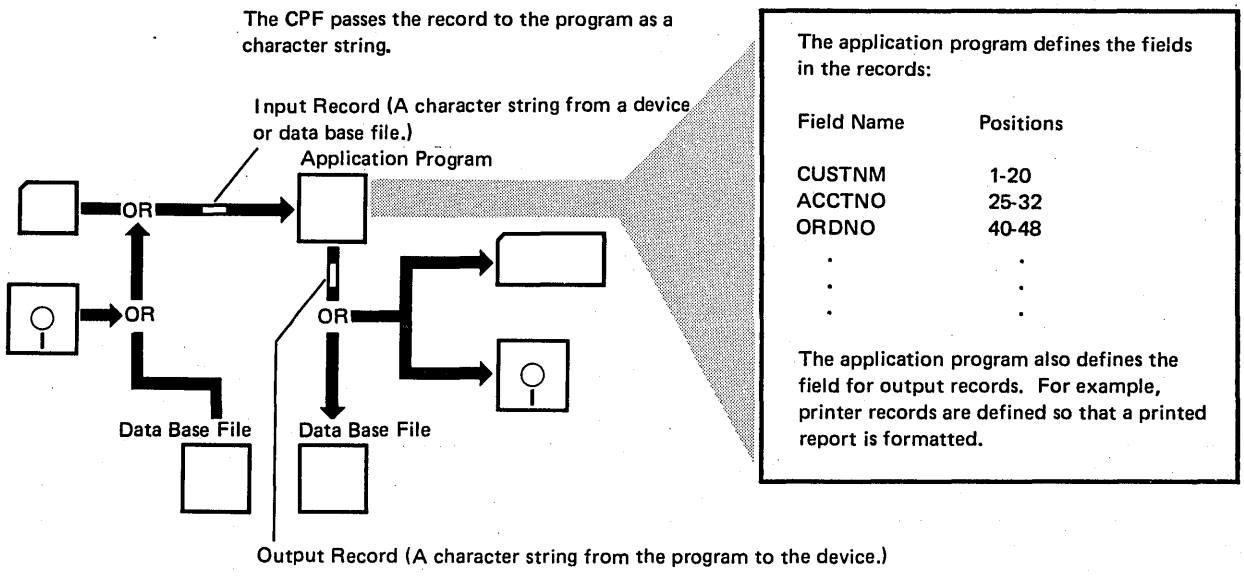
DATA OPERATIONS

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, for data base files, any data in storage that is associated with it are 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:

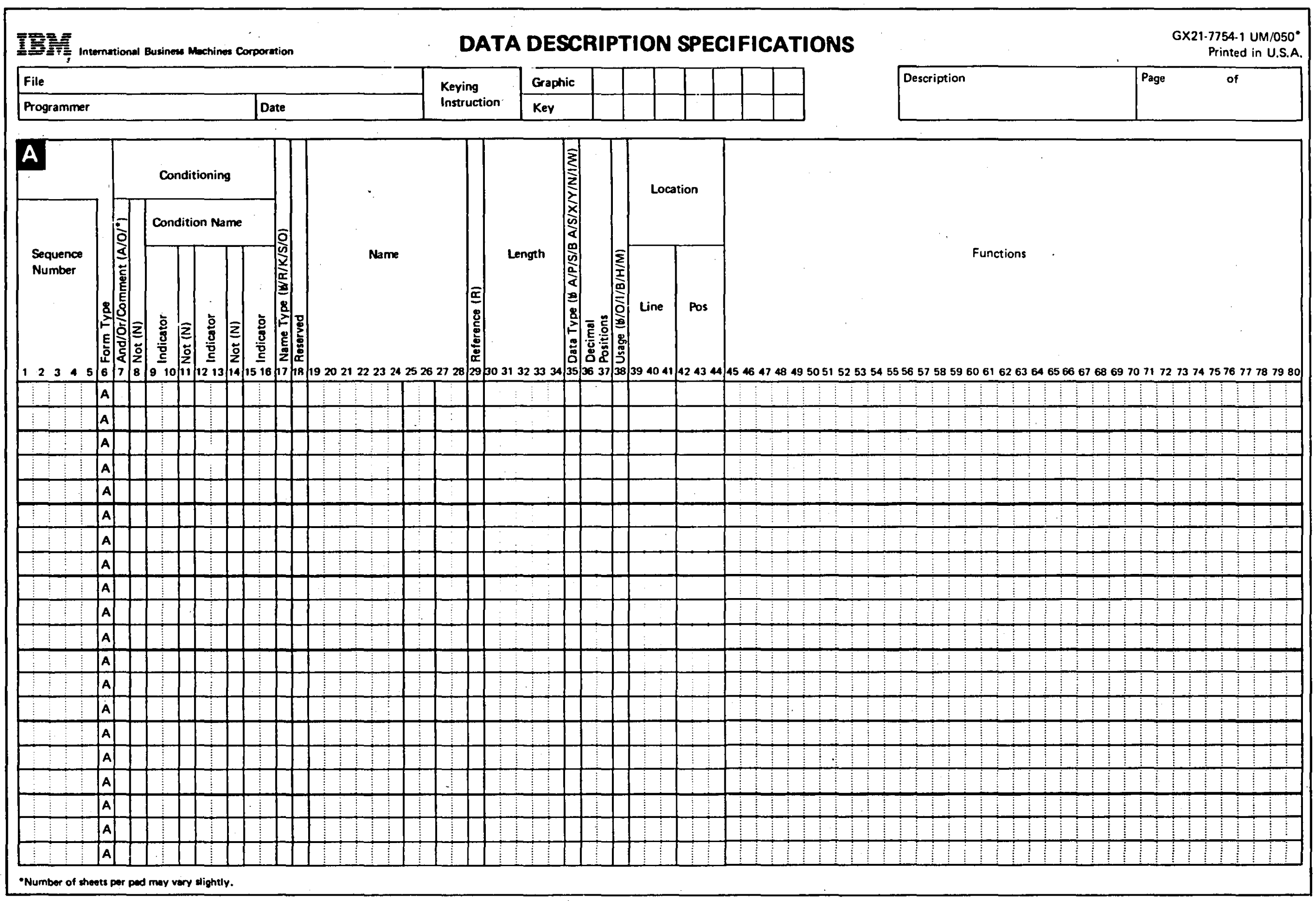


Externally Described Data Files

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 to change device files. Deleting a file destroys the file's description and any data in storage that is associated with it.

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

GX21-7754-1 UM/050*
Printed in U.S.A.

File | Keying Instruction | Graphic | Key | Description | Page of
Programmer | Date

A

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

(Conditioning: Condition Name)

*Number of sheets per pad may vary slightly.

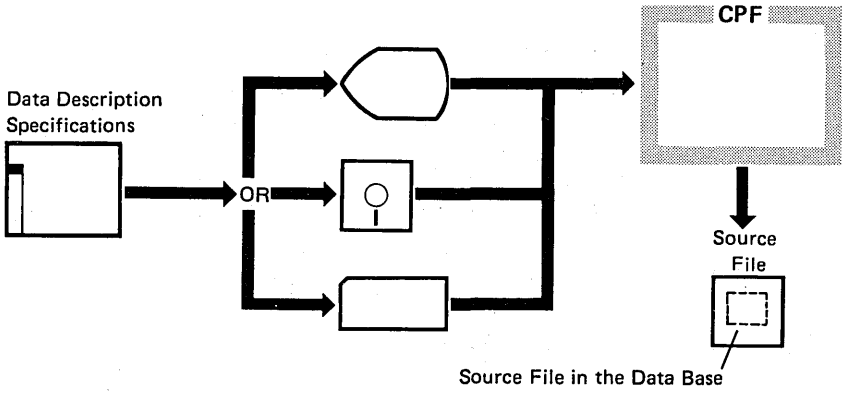
The form is used for both data base files and device files. A complete description of the data description form and the entries allowed for each position is contained in the *CPF Reference Manual—DDS*. Examples showing how the form is used are contained in the *CPF Programmer's Guide*.

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

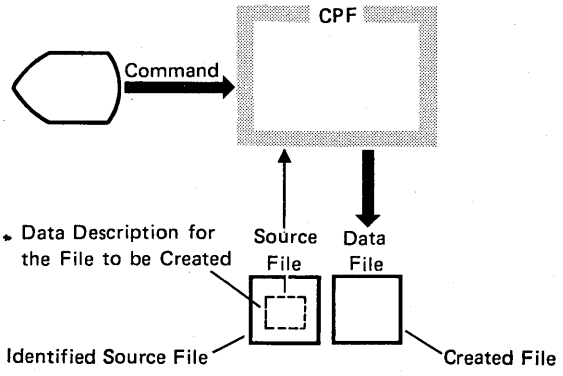
- Use a card or diskette source file.
- Use the source entry utility (SEU), a part of the Interactive Data Base Utilities, to build a source file in the data base.

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. CPF provides a listing of the source statements used to create the file. 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 the //DATA command, which does not specify a file name. Once an unnamed file has been processed by the job, it cannot be accessed again in the job. 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 also identified in the spooled input by the //DATA command, which 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 available until the job ends.

Using Inline Data Files:

In the following example, an unnamed inline data file is included in the input stream. This file, identified by the // DATA command, can be used by any of the programs in the first job. However, it can be opened and used only once. (If an inline data file is to be accessed more than once by the job, a named, inline spool file must be used.)

```
// JOB DAILYSALES JOBPTY(2)
RPLLIBL  LIBL(ORDLIB QGPL QTEMP)
CALL  SALES205                    /*PRINT DEPT SUMMARY*/
CALL  SALES206                    /*PRINT PRODUCT SUMMARY*/
CALL  SALES209                    /*PRINT SPECIAL SALES*/
// DATA
      AJ1052
      XQ4031
      BZ9504
        .
        .
        .
CALL  SALES213
// ENDJOB
// JOB DAILYSHIPT JOBPTY(5)
        .
        .
        .
// ENDJOB
```

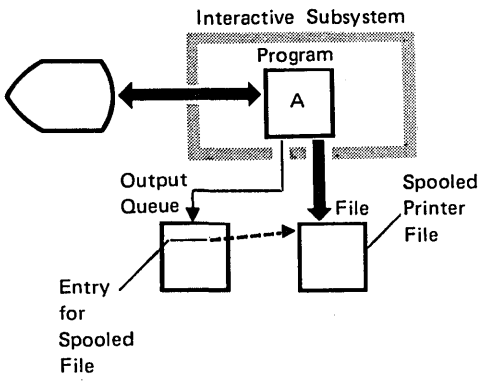
Spooled Output Files

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:

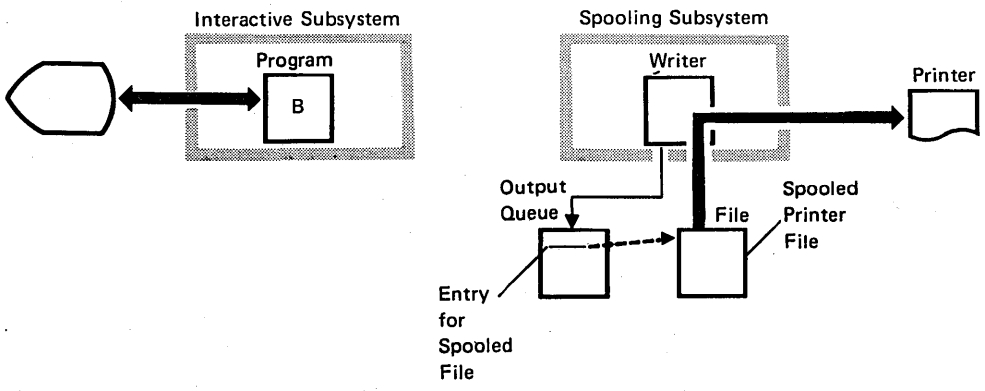
- The output queue for the file
- The type of forms to be used
- The number of copies to be produced
- The maximum number of records that can be placed in the file
- Whether the spooled file can be written to the device while the program is still producing output
- Whether the file should be saved on 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 appropriate output queue and the data is placed in a spooled file in the system. In the following drawing, program A is executing in the interactive subsystem. The program produces a spooled printer file. As the program executes, the records for the printer file are placed in the spooled file in the system.



The spooled file can be made available for printing when the file is opened, when the file is closed, or at the end of the job. As shown in the following drawing, a writer is started in the spooling subsystem to write the records to the printer. The output for the file is selected from an output queue. This can occur while program A is active or, as shown here, when program A is no longer active and program B is active.



Copying Files

System/38 provides functions for copying data from one data base file or device file 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 (except to tape and display device files)
- Copy from a device file to another device file
- Copy from a device file to a physical data base file

Records can be added to the receiving file, or they can completely replace any previously existing records in the receiving file.

Both externally described data files and program-described data files can be copied. Through the use of the copy functions, some operations that could previously be performed only by application programs can be performed. These operations include:

- Selecting only a subset of records from the file being copied
- Omitting a portion of the 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 (externally described data files only)
- Changing the format of the records as they are copied by deleting or adding fields to the record formats of the new file (externally described data files only)

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, could be used to examine specific information in the data base file to find such information as:

- Which programs use a particular file
- Which files contain record formats that use a particular field

Chapter 5. Application Development

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 many 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 task that ties up a work station for more than a short time (while the task executes and the work station user cannot interact with the system) should be executed as a batch job.

In System/38, batch jobs can be submitted by both the system operator and other work station users. 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 to 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 (*NOLIST *LIST)


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.

User-defined menus can be designed through display device files. These menus, which can be tailored specifically to the needs and experience of the work station users, can provide a specialized user interface through which application programs can be invoked. The following menu is an example of a user-defined menu that could be used for an order entry application.

```
              ORDER DEPT CONTROL CLERK MENU
   Select one of the following:
     1. General Menu
     2. File Maintenance Menu
     3. Batch Job Menu
     9. Sign off


Option: _________
```

The work station user selects an option to display another menu from which he can select the option that starts his program. Of course, in smaller applications, the first menu the work station user responds to could start the program.

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. In this case, 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 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 in Chapter 4, *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 in Chapter 4, *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 can pass information from one program to another in one of the following ways:

- 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 base file
- In a data area object

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 files to be created, source files containing the source statements are needed. The user can place source statements in the data base by copying them from a device file or by entering the source statements through the source entry utility, which is part of the Interactive Data Base Utilities. 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 (SEU) provides special display formats to help the user enter specifications for other programs and for 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 *SEU 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 maintaining 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. There are many advantages in using control language programs in an application. 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 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 between 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.

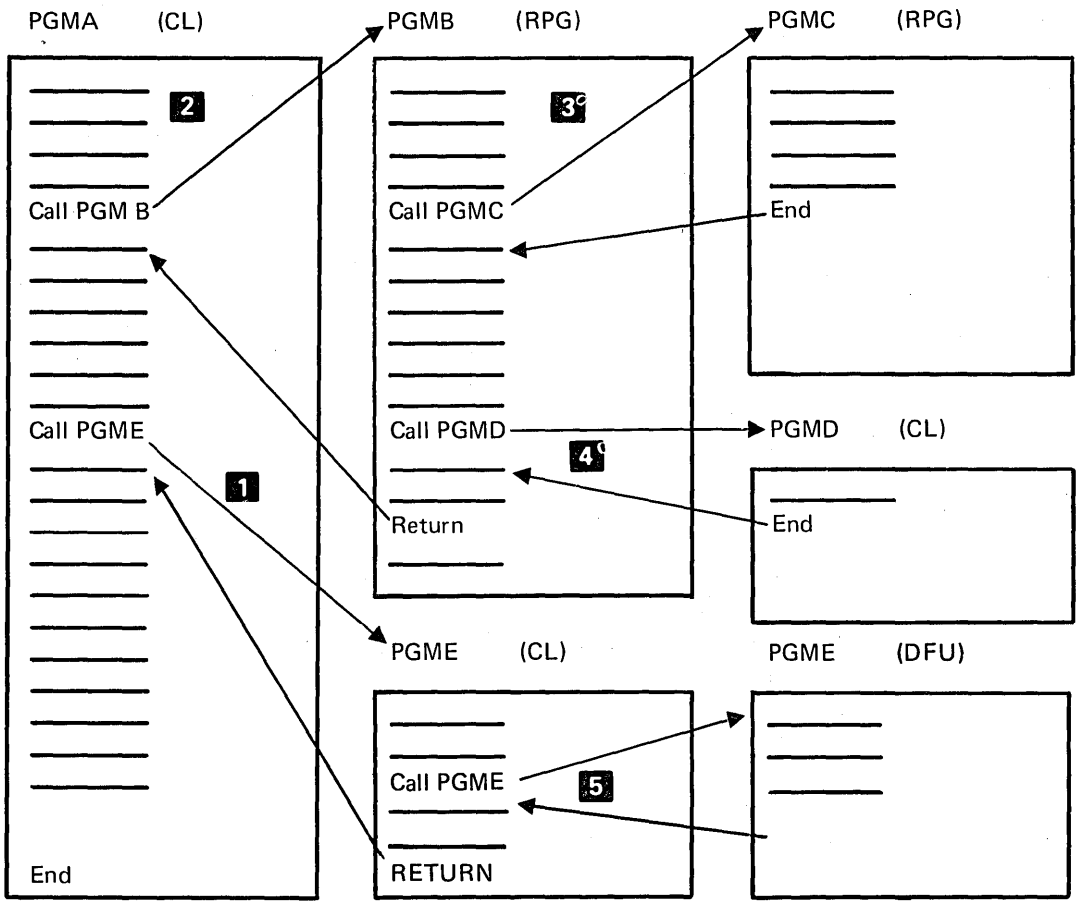
- Provide system functions that are not available through other high-level languages.

- 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, RPG programs, and DFU (data file utility) 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:

1. A control language program calling another control language program
2. A control language program calling an RPG program
3. An RPG program calling another RPG program
4. An RPG program calling a control language program
5. A control language program calling a DFU program



MESSAGE HANDLING

A *message* is a communication sent from one user or program to another. Most data processing systems provide communication between the system and the system operator to handle errors and other conditions that occur during processing. In addition to this type of support, CPF provides message handling functions that support two-way communications between programs and system users, between different programs, and between different system users. The CPF message handling functions support the use of:

- Impromptu messages, which are created by the program or system user when they are sent and are not permanently stored in the system.

- Predefined messages, which are created before they are used. These messages are placed in a message file when they are created and retrieved from that file when they are used.

Because messages can be used to provide communication between programs and between programs and system users, the use of the CPF message handling functions should be considered when applications are developed. The following concepts of message handling are important to application development:

- Messages can be defined in message files, which are outside the programs that use them, 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 send messages to a user who requested the program regardless of what work station that user has signed on to. Messages do not have to be sent to a specific device; thus, one program can be used from different work stations without change.

Because replies can be returned by a user or a 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 the 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 the user. The program or the user obtains the message by receiving it from the queue.

CPF provides message queues for:

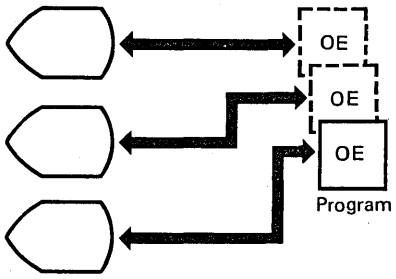
- Each work station on the system
- Each job on the system and each active program within a job
- The system operator
- The system logs

Additional message queues can be created to meet any special application requirements. Messages sent to message queues 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 decides to receive them.

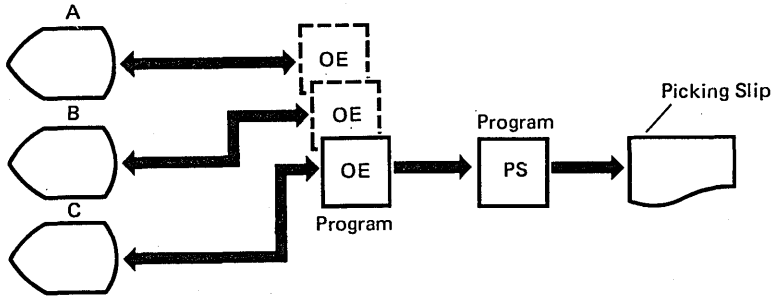
Using Messages and Message Queues

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:

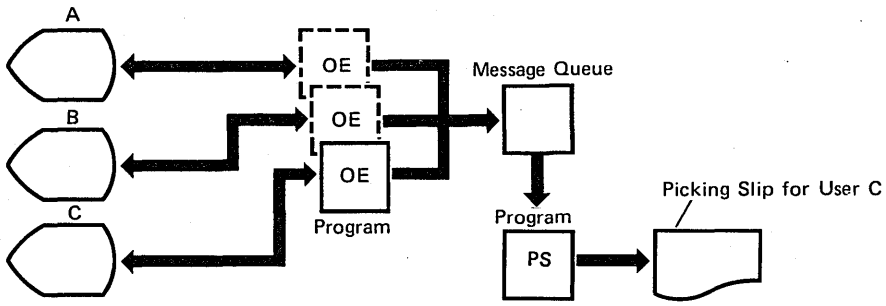
1. Three work station users enter orders using the same order entry program. (Because of the design of the system, all three users actually share a single copy of the executable program.)



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. Message processing, as shown here, is supported through control language programs.



DEBUGGING FUNCTIONS

CPF includes functions that let a programmer observe operations that are performed as a program executes. These functions can be used to locate operations in the program that are not performing as intended. Debugging functions can be used either in batch jobs or interactively from a work station. In either case, the program being observed must be 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 always have to check the parameters passed to it.

The command definition functions can be used to:

- Create unique commands needed by an installation or individual users.
- Define alternative versions of commands provided by CPF to meet the requirements of an installation or individual users. This function might include having different 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. The IBM-supplied commands should not be changed.

More detailed information on command definition is contained in the *CPF Programmer's Guide* and *CPF Reference Manual—Control Language.*

Chapter 6. System Management

The System/38 CPF provides the facilities that are needed to regulate the use and the 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.

Security and authorization functions provided by the System/38 can reduce the risk of unauthorized use, but will not eliminate it. For these functions to be effective, proper user implementation should be accompanied by other control practices, such as physical security and division of duties. Overall controls and security are the responsibility of the system security officer.

For many data processing installations, maintaining the integrity and the security 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.

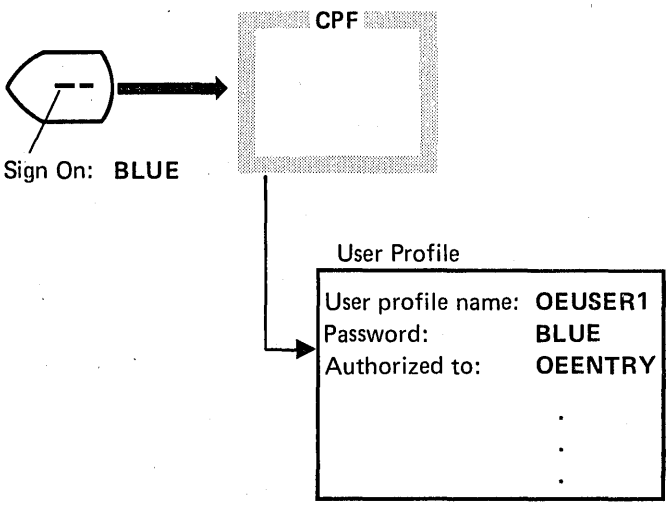
User Identification

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 and can also identify a program (called an initial program) that is to be executed when the user signs on the system. This initial program is called only if the user's job is routed to the command language processor (program QCL). 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.

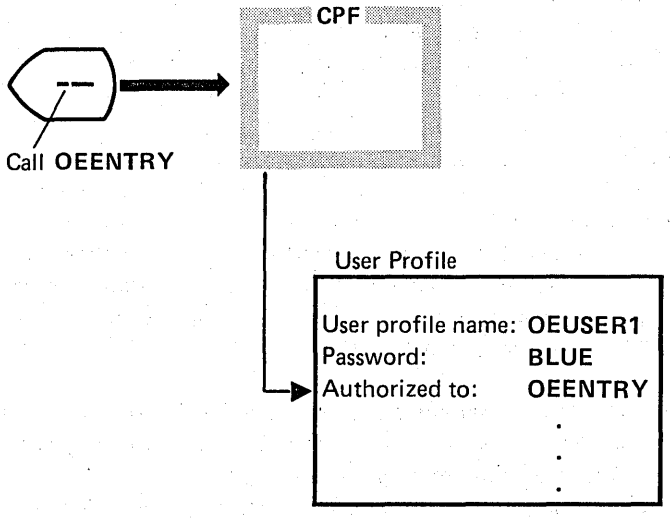
When a work station user signs on, his password is used by CPF to find the user's profile. Each user profile is named. The user is known in the system by the name of his user profile; thus, references to a user in the system do not need to be changed when a password is changed. The following drawing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

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

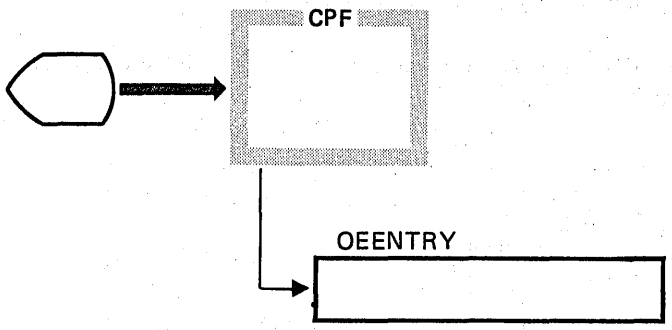
1. The CPF uses the password (BLUE) to determine which user profile to use.



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



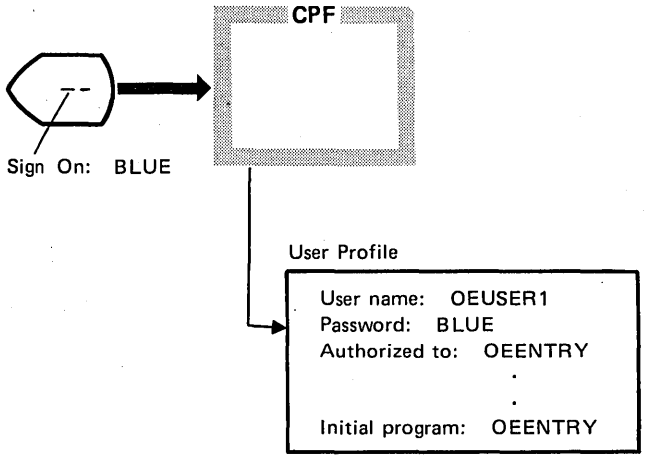
3. If the user is authorized to use the object, the request is honored.



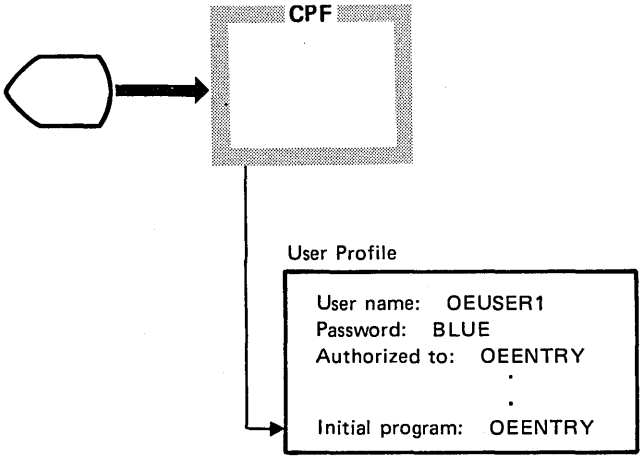
In this example, the user (OEUSER1) calls the program OEENTRY. If that program is to be executed whenever OEUSER1 signs on the system, OEENTRY could be specified as the initial program in the user profile. In this case, the example proceeds as follows.

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

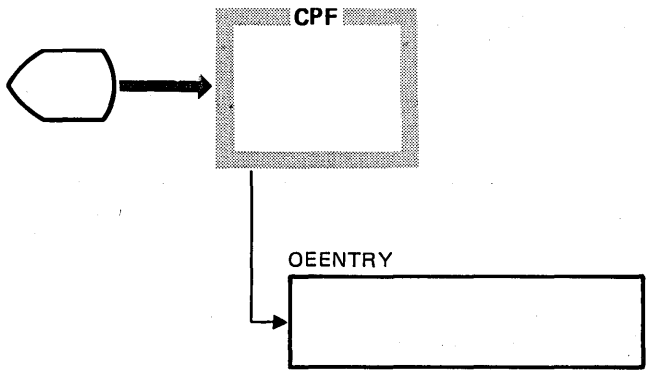
1. CPF uses the password to determine which user profile to use.



2. CPF, through the work management facilities, initiates the job and determines from the user profile that the program OEENTRY is to be invoked for the job.



3. The program is started and the work station user interacts with the program OEENTRY.



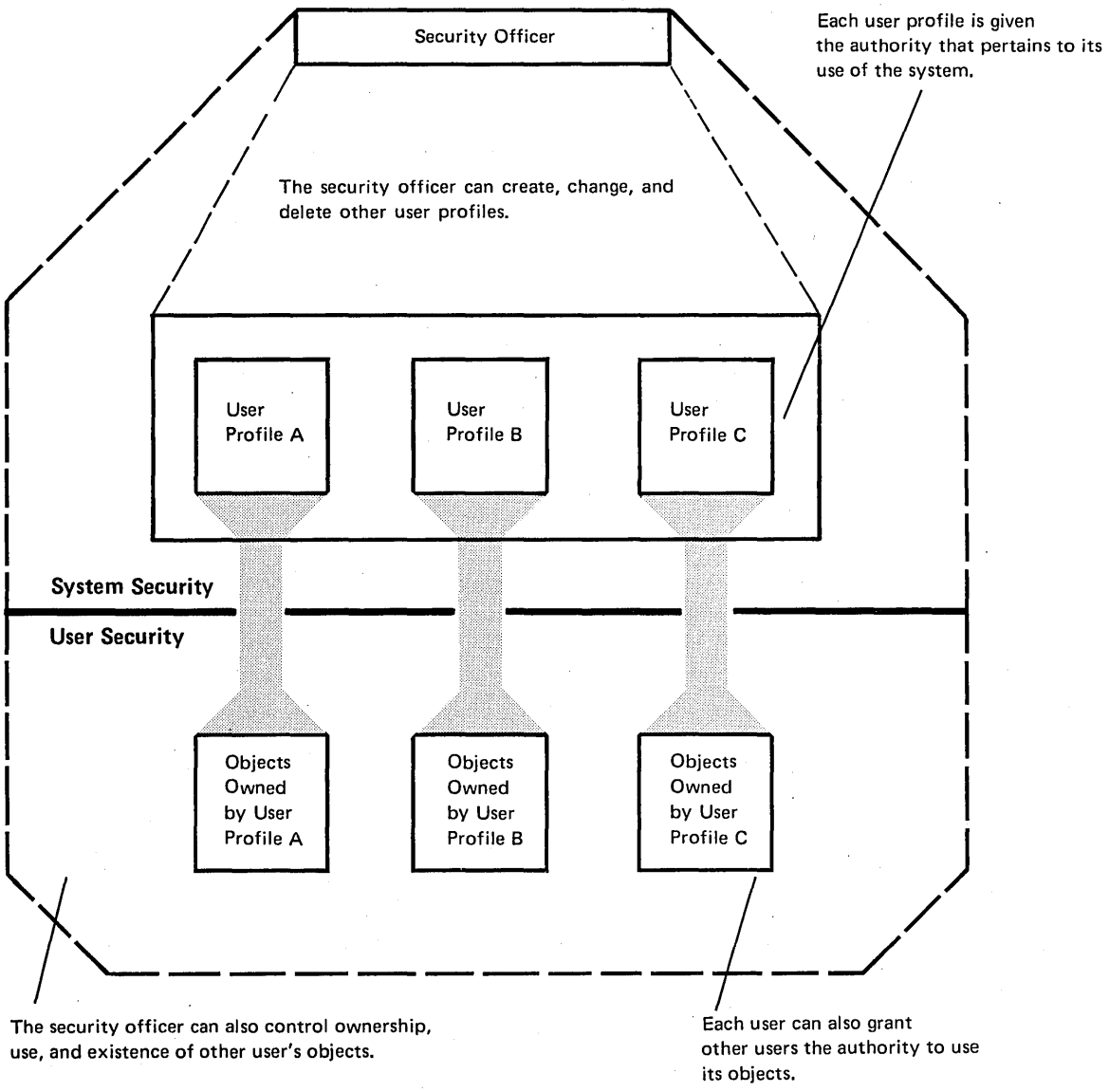
In this example, the work station user does not interact with CPF after he has signed on the system. Instead, the initial program provides the interface between the user and the system. This interface can be a menu or prompt that is especially designed for the work station user's needs and experience.

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 a 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 to execute application programs
- The program support representative: Lets service personnel maintain CPF
- The customer engineer: Lets service personnel use a set of functions called the concurrent service monitor to maintain the machine

Security Functions

Because all the functions and the 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 a user's authority 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

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 give certain users the authority only to read data, while other users may be given authority to read and update data. 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 outside the system and later restore them to the system. These save/restore facilities 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 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 storage outside the system 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 that occur during system operation.

Save Functions

The save functions write a copy of an object onto diskettes or tape. 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 can be 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 a file or program 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 objects back into the system. These functions are used to restore any saved object. The system library objects are copied back into the system by the CPF installation and specialization facilities described later 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 work stations, the control units, and the lines on the system to CPF
- Creating any unique print images and translate tables needed by the application programs
- Creating user-defined libraries
- Changing system values to meet specific system requirements
- 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
- Creating new edit descriptions

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

CPF provides the following menu to assist the system operator:

```
                    SYSTEM OPERATOR MENU
Select one of the following:
  1. DSPJOBQ (jobq)
  2. DSPOUTQ (outq)
  3. SNDMSG  tomsgq,(type),msg
  4. CALL    program
  5. Execute command
  6. SBMJOB  (job),(jobd),(cmd)
  7. STRPRTWTR  device,outq
  8. CNLWTR     writer
  9. STRDKTRDR  device,label
 10. CNLRDR     reader
 11. SIGNOFF    (*NOLIST *LIST)
 Option: ___ Parms: _____________________________________________
 Msg or cmd: ____________________________________________________
_________________________________________________________________
 Log requests: *YES
 CF3-Command entry  CF6-DSPMSG QSYSOPR  CF7-DSPSBS  CF8-DSPSYS
```

When the system operator uses this menu, he can perform frequently used tasks quickly and easily. He can perform many common tasks by simply selecting an option from the menu. The operator can request functions that are not included on the menu by selecting option 5 and entering the appropriate control language command. Many of the functions selected from the menu require fewer system resources than they would if commands were entered to request them.

Many of the operations that are traditionally performed by the system operator on other systems can also be performed by other work station users on System/38. These operations include submitting batch jobs to a job queue, starting and terminating spooling readers and writers, and starting and terminating subsystems. The use of these functions can be limited through each user profile.

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. The CPF message-handling functions support the use of:

- 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, in one operation, one message to all the work stations.

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 can be 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.

Glossary

access path: The means by which CPF 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 or the system that specifies the maximum number of jobs that can execute concurrently in the storage pool or the system.

allocate: To assign a resource for use in performing a specific task. Contrast with *deallocate.*

application: A particular data processing task, such as an inventory control application or a payroll application.

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

attribute: A characteristic; for example, attributes of a field include its length and data type, and attributes of a job include its user name and job date.

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

autostart job: A job that is automatically initiated when a subsystem is started.

autostart job entry: A work entry in a subsystem description that specifies a job to be automatically initiated each time the subsystem is started.

auxiliary storage: All addressable storage other than main storage. Auxiliary storage is located in the system's nonremovable disk enclosures.

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.

batch processing: A method of executing a program or a series of programs in which one or more records (a batch) is processed with minimal or no interaction with the user or operator. Contrast with *interactive processing.*

class: An object that contains the execution parameters for a routing step. The system-recognized identifier is *CLS.

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

command definition: An object that contains the definition of a command (including the command name, parameter definitions, and validity checking information) and identifies the program that performs the function requested by the command. The system-recognized identifier is *CMD.

command processing program: A program that processes a command. This program performs some validity checking and executes the command so that the requested function is performed. Abbreviated CPP.

control language: The set of all commands with which a user requests functions. Abbreviated CL.

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

Control Program Facility: The system support licensed program for the IBM System/38. It provides many functions that are fully integrated in the system such as work management, data base data management, job control, message handling, security, programming aids, and service. Abbreviated CPF.

controlling subsystem: An interactive subsystem that is started automatically when the system is started and through which the system operator controls the system. IBM supplies one controlling subsystem: QCTL.

CPF: See *control program facility.*

data base: The collection of all data files stored in the system.

data base file: An organized collection of related records in the data base. See also *physical file* and *logical file.*

data description specifications: A description of the user's data base or device files that is entered using a fixed-form syntax. The description is then used to create files. Abbreviated DDS.

data file utility: The utility of the Interactive Data Base Utilities licensed program that is used to create, maintain, and display records in a data base file. Abbreviated DFU.

data rights: The authority to read, add, update (modify), or delete data contained in an object.

deallocate: To release a resource that is assigned to a specific task. Contrast with *allocate*.

device description: An object that contains information describing a particular device that is attached to the system. The system-recognized identifier is *DEVD. Abbreviated DEVD.

device file: An object that describes how the data is processed on an external input or output device attached to the system, such as a work station, a card device, a printer, the diskette magazine drive, a magnetic tape drive or a communications link.

DFU: See *data file utility*.

display file: A device file associated with a display work station or console that describes the content of one or more displays.

externally described data: Data contained in a file for which the fields in the records are described to CPF, by using data description specifications, when the file is created. The field descriptions can be used by the program when the file is processed. Contrast with *program-described data*.

externally described data file: A file containing data for which the fields in the records are described to CPF, by using data description specifications, when the file is created. The field descriptions can be used by the program when the file is processed. Contrast with *program-described data file*.

file: An object that contains a description of a set of related records treated as a unit and, optionally, those records. The system-recognized identifier is *FILE.

file description: The information contained in the file that describes the file and its contents.

file overrides: The file attributes that can be specified at execution time that will override the attributes specified in the file description or in the program.

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

high-level language: A programming language that relieves the programmer from the rigors of machine level or assembler level programming; for example: RPG III and COBOL. Abbreviated HLL.

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 Data Base Utilities: A System/38 licensed program that consists of DFU, SEU, query, and SDA. Abbreviated IDU.

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

interactive subsystem: A subsystem in which interactive jobs are to be processed. IBM supplies three interactive subsystems: QCTL, QINTER, and QPGMR.

invocation: The execution of a program.

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 that contains the attributes of a job. The system-recognized identifier is *JOBD.

job priority: The order in which batch jobs on a job queue are to be processed. More than one job can have the same priority.

job queue: An object on which batch jobs are placed when they are submitted to the system and from which they are selected for execution by CPF. The system-recognized identifier is *JOBQ.

job queue entry: A work entry in a subsystem description that specifies the job queue from which the subsystem can accept batch jobs.

key field: A field of a record whose contents are used to sequence the records of a particular type within the file.

keyed-sequence access path: An access path to a data base file that is based on the contents of key fields contained in the individual records.

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. The system-recognized identifier is *LIB.

library list: An ordered list of library names used to find an object. The library list indicates which libraries are to be searched and the order in which they are to be searched. The system-recognized identifier is *LIBL. *LIBL specifies to the system that a job's current library list is to be used to find the 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. Contrast with *physical file.*

main storage: All storage in a computer from which instructions can be executed directly.

member: 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 person or program.

message description: The descriptive information about a message and the text of the message.

message queue: An object on which messages are placed when they are sent to a person or program. The system-recognized identifier is *MSGQ.

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

object authority: The right to use or control an object. See *object rights* and *data rights.*

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, they are all fixed-length records and they all contain the same fields in the same order. Contrast with *logical file.*

production library: A library containing objects needed for normal processing. Contrast with *test library.*

program-described data: Data contained in a file for which the fields in the records are described in the program that processes the file. Contrast with *externally described data.*

program-described data file: A file for which the fields in the records are described only in the program that processes the file. Contrast with *externally described data file.*

public authority: The authority to an object granted to all users.

qualified object name: An object name and the name of the library containing the object.

query: A request to extract, from a file, one or more records based upon a criterion.

queue: A line or list formed by items in the system waiting for service; for example, work to be performed or messages to be displayed.

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

record: An ordered set of fields that make up a single occurrence of a record format. A record is the basic unit of data transferred between a file and a program.

record format: The definition of how data is structured in the records contained in a file. The definition includes the record name and field descriptions for the fields contained in the record. The record formats used in a file are contained in the file's description.

restore: To transfer specific objects or libraries from magnetic media such as diskettes or tape to internal storage by duplicating them in internal storage. Contrast with *save*.

routing data: A character string that CPF compares with character strings in the subsystem description routing entries to select the routing entry that is to be used to initiate 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 a routing step that executes in the subsystem.

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

save: To transfer specific objects or libraries from internal storage to magnetic media such as diskettes or tape by duplicating them on magnetic media. Contrast with *restore*.

security: The control of access to, or use of, data or functions.

security officer: The individual at an installation who is designated to control the authorization of functions and data in the System/38.

single level storage: The technique of addressing multiple levels of storage through a single addressing structure.

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 provides access to data processed by readers or causes output data to be saved for later processing by a writer.

spooling: The CPF-provided execution-time support that reads and writes input and output streams on an intermediate device in a format convenient for later processing or output.

spooling subsystem: A subsystem that provides the operating environment needed by the CPF programs that read jobs onto job queues and write files from the output queues. A subsystem description for a spooling subsystem is provided as part of the CPF and is named QSPL.

storage pool: A logical segment of main storage reserved for executing a group of jobs.

subsystem: A predefined operating environment through which CPF 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 CPF uses to control the subsystem. The system-recognized identifier is *SBSD.

system library: The library provided by CPF to contain system-oriented objects provided as part of CPF. Named QSYS.

system operator: The individual who operates the system and looks after the peripheral equipment necessary to initiate computer runs or finalize the computer output in the form of completed reports and documents.

temporary library: A library that is automatically created for each job to contain temporary objects that are created by that job. The library is deleted when the job ends. Named QTEMP.

test library: A library to be used in debug mode and that does not contain objects needed for normal processing. Contrast with *production library*.

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 contains a description of a particular user or group of users. A user profile contains a list of authorizations to objects and functions. The system-recognized identifier is *USRPRF.

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

work station: A device that lets a person transmit information to or receive information from a computer as needed to perform his job. In System/38 publications, work station refers to a device that has a screen and keyboard.

work station entry: A work entry in a subsystem description that specifies the work stations from which users can sign on to the subsystem or from which interactive jobs can transfer to the subsystem.

work station user: A person who uses a work station to communicate with the System/38.

Index

access groups 1-8
access paths
  arrival sequence 4-8
  definition of G-1
  general description 4-8
  keyed sequence 4-9
  sharing 4-13
access to system, controlling 6-1
activity level 3-7
activity level control 1-8
adding routing entries 3-18
adding work entries 3-18
adopt user profile 6-10
allocate G-1
allocating objects 3-20
allocation of storage, objects 2-2
application G-1
application design 5-1
application development
  description of 5-1
  overview 1-4
application structure 5-5
application, structuring 5-5
area, data 5-5
arrival-sequence access path G-1, 4-8
ascending sequence 4-9
asynchronous job execution 3-22
attribute G-1
attributes
  display field 4-22
  file 4-2
  job 3-11
  objects 2-1
authority G-1
authority, object 6-7
authorization, objects 6-7
authorizing system users 6-7
autostart job G-1
autostart job entry G-1, 3-10
auxiliary storage G-1

backing up the system 6-9
batch applications, design 5-1
batch job
  definition G-1, 3-3
  initiating 3-21
batch processing G-1
batch subsystem 3-17

call menu, program 5-2
canceling jobs 3-19
card device support 4-28
changing job attributes 3-19
changing passwords 6-7
changing routing entries 3-18
changing subsystem descriptions 3-18
changing work entries 3-18
characteristics, device 4-19
checking, record format level 4-5
class G-1, 3-13
class operations 3-18
clearing a library 2-8
command
  definition G-1, 5-15
  definition of G-1
  description of 1-9
  entry display 1-10
  name, description of 1-9
  parameters 1-10
  processing program G-3, 5-15
  prompting 1-10
  syntax 1-9
communication, program-user 5-3
concepts
  data base 4-7
  data management 4-1
  work management 3-2
concepts, system 1-5
concurrent service 6-14
conditioning indicator 4-23
connecting a file to a program 4-6
consecutive record processing 4-11
control language
  definition of G-1
  general description 1-9
  logic functions 5-9
  programs G-1, 5-8
control program facility, definition of G-1, 1-1
controlling resource usage 3-2
controlling subsystem G-1, 3-4, 3-17
controlling system operation 1-2
controlling work flow 3-2
copy operation 4-36
copying files 4-36
CPF
  definition of G-1, 1-1
  interfaces 1-2
  overview 1-2
CPF-provided libraries 2-5
CPF-provided subsystems 3-16
CPF-provided user profiles 6-6

creating
edit descriptions 6-11
libraries 2-8
print images 6-11
programs and files 5-6
subsystem descriptions 3-18
customer engineer, user profile 6-6

data area 5-5
data association specifications 4-2
data base
definition of G-1, 4-7
file design 5-4
functions 1-8
data base data management 4-7
data base file
definition of G-1, 4-1
using 4-18
data description specifications
definition of G-2, 1-3
form 4-30
source statements 4-31
use of 4-2
data file utility G-2, 5-10
data files
designing 5-4
inline 4-32
data integrity 6-1
data management
card device 4-28
concepts 4-1
data base 4-7
device support 4-19
diskette 4-28
facilities 4-1
overview 1-3
printer 4-27
data operations
externally described data files 4-30
general description 4-29
program described data files 4-29
data path 4-6
data portion, objects 2-1
data rights G-2, 6-8
data, description of 4-2
data, routing 3-9
deallocating objects 3-20
debugging and testing 5-6
debugging functions 5-14
defaults, parameter 1-11
defining commands 5-15
defining devices 6-11
definitional objects, subsystem 3-14
deleting
libraries 2-8
subsystem descriptions 3-18
dependent programs, file 4-7
descending sequence 4-9

describing data 4-2
description
device 4-19
file 4-2
job 3-11
logical file 4-18
message 5-12
physical file 4-11
text 5-7
design considerations, application 5-1
designing data files 5-4
developing applications 5-1
device
characteristics 4-19
configuration 6-11
description G-2, 4-19
file design 5-4
file, definition of G-2
files 4-19
files, definition of 4-1
device support
card 4-19, 4-28
communications 4-19, 4-27
data management 4-19
diskette magazine drive 4-19, 4-28
display 4-19, 4-20
nondisplay 4-27
printer 4-27
tape 4-19, 4-28
DFU (data file utility) G-2, 5-10
diskette magazine drive support 4-19, 4-28
display
command entry 1-10
device support 4-20
fields 4-23
file descriptions 4-21
file record formats 4-22
file, definition of G-2
files 4-20
library contents 2-8
prompt 1-11
subsystem descriptions 3-18
displaying
jobs 3-19
object descriptions 2-7
subsystem status 3-18
documentation 5-7

edit descriptions, creating 6-11
enrolling users 6-7
entering commands 1-9
entering source statements 5-6
entries
routing 3-9
work 3-7
work station 3-7
entry, job queue 3-8
environment, operating 3-4

exclusive file allocation 4-6
execution control operations 3-19
execution environment, machine 3-13
explicit file allocation 4-6
external device support 4-19
externally described data G-2, 4-2
externally described data file
  creating 4-30
  definition of G-2
  operations 4-30

facilities
  data management 4-1
  installation and specialization 6-11
  object management 2-1
  service 6-14
  work management 3-2
field level description 4-2
fields, display 4-22
file
  connecting to program 4-6
  definition of G-2, 4-1
  members 4-11
  opening 4-6
  overrides G-2, 4-6
  processing 4-6
  processing, spooled 4-32
  source 4-5
file attributes
  general description 4-2
  special 4-5
file dependent programs 4-7
file description
  definition of G-2
  display file 4-21
  general description 4-2
  logical file 4-16
  physical file 4-11
file independent programs 4-6
file reference function 4-37
file usage, tracking 4-37
files
  card 4-28
  copying 4-36
  creating 4-31, 5-6
  data base 4-7
  designing 5-4
  diskette 4-28
  display 4-20
  general description 4-1
  message 5-11
  physical 4-11
  printer 4-27
  spooled 4-5
finding objects in libraries 2-5
formatting, screen data 4-20
free storage 6-9

functions
  debugging 5-14
  restore 6-10
  save 6-10
  save/restore 6-9
  security 6-7
  system operation 6-12

general object operations 2-7
general purpose library G-2, 2-5
generic keys 4-9
granting authority 6-7
grouping objects 2-2

handling messages 5-11
high-level language G-2, 4-28
history information, save/restore 6-10
holding a job 3-19

identification, user 6-1
identifier, record format level 4-5
identifying objects 2-2
impromptu messages 6-13
independent programs, file 4-6
information, passing 5-5
initiating jobs 3-21
inline data file G-2, 4-32
input spooling 4-32
installation and specialization
 facilities 6-11
integrity, data 6-1
integrity, definition of G-2
interactive
  application, design 5-1
  command prompting 1-10
  debugging 5-14
  job, definition of G-2, 3-2
  jobs, initiating 3-21
  subsystem G-2, 3-17
Interactive Data Base Utilities G-2, 5-6
interface, user 5-2
interfaces to CPF 1-2
invocation, definition of G-2
invoking an application 5-2

job
  definition of G-2, 3-2
  general description 3-11
  holding 3-19
  transferring 3-20
job attributes, changing 3-19
job description
  definition of G-2
  general description 3-11
  operations 3-19
job entries, autostart 3-10
job priority
  batch 3-21
  definition of G-2
job priority, batch 3-21
job queue entry G-2, 3-8
job queue, definition of G-3, 3-3
job stream, example 3-23
job/subsystem relationships 3-14
jobs
  batch 3-3
  canceling 3-19
  displaying 3-19
  general description 3-10
  holding 3-19
  initiating 3-21
  interactive 3-2
  managing 3-19
  releasing 3-19
  rerouting 3-20
  submitting 3-19, 3-22

key field G-3, 4-9
key field, definition of 4-9
key, generic 4-9
keyed-sequence access path G-3, 4-9
keyed sequence files, processing 4-9
keyword parameters 1-10

level checking, record format 4-5
level identifier 4-3
libraries
  backing up 6-10
  CPF-provided 2-5
  general description 2-2
  test 5-6
library list
  definition of G-3, 2-5
  use 2-6
library operations 2-8
library search 2-5
library types 2-5
library, definition of G-3, 2-2
list, library 2-5
locating objects 2-5
logic functions, control language 5-9
logical file G-3, 4-12
logical file description 4-18

machine execution priority 3-13
main storage
  definition of G-3
  pools 3-7
management facilities
  data 4-1
  object 2-1
  work 3-2
managing jobs 3-19
managing libraries 2-8
managing subsystems 3-18
managing the system 6-1
member G-3, 4-11
menu, program call 5-2
message
  definition of G-3
  descriptions G-3, 5-12
  files 5-11
  handling
    general description 5-11
    system operation 6-12
  queue G-3, 5-12
  text 4-23
messages, using 5-13
monitor, subsystem 3-14
monitoring system operation 6-12
moving objects 2-2, 2-7
multifunction card unit support 4-28
multiple-field keys 4-9
multiple record formats 4-14

named inline data files 4-32
names, object 2-2
nondisplay device support 4-27
normal object authority 6-8

object
  allocating 3-20
  attributes 2-1
  authority G-3, 6-7
  damage 2-9
  data portion 2-1
  deallocating 3-20
  definition of G-3, 1-2, 2-1
  description, displaying 2-7

object (continued)
finding 2-5
identification 2-2
management concepts 2-1
management facilities 2-1
management operations 2-7
management, overview 1-2
moving 2-2, 2-7
name, qualified 2-2
names 2-2
operations
general 2-7
job 3-19
organization 2-2
ownership 6-7
renaming 2-7
restoring 6-10
rights G-3, 6-7
save/restore operations 2-7, 6-9
saving 6-10
security operations 2-7
types 2-1
object-oriented architecture 1-6
object use, rights of 6-7
objects in a library 2-2
opening a file 4-6
operating environment 3-4
operational characteristics, object 2-2
operations
data 4-29
library 2-8
object 2-7
subsystems 3-18
system 6-10
operator, system 6-5
organizing objects 2-2
output files, spooled 4-34
output spooling 4-34
overrides, file 4-6
overriding job attributes 3-12
ownership, object 6-7

parameter defaults 1-11
parameters, definition of 1-10
passing parameters 5-5
password
changing 6-7
definition of G-3
general description 6-3
path, access 4-8
physical file G-3, 4-11
physical file record format 4-11
pools, storage 3-7
predefined job 3-2
predefined messages 6-13
predefined operating environment 3-4
predefined user profiles 6-6
print images, creating 6-11
printer file support 4-27
priority
job queue 3-21
machine execution 3-13
processing a file 4-6
processing keyed sequence files 4-9
processing spooled files 4-32
processor time 3-13
profile, user 6-2
program
command processing 5-15
connecting to a file 4-6
control language 5-8
creating 5-6
file dependent 4-7
file independent 4-6
program call menu 5-2
program-described data G-3, 4-2, 4-4
program described data files, operations 4-29
program invocation 5-2
program support representative, user profile 6-5
program-user communication 5-3
programmer user profile 6-4
programming considerations 5-5
prompts, commands 1-10
protected environment, testing 5-6
public authority G-3, 6-7

qualified object name G-3, 2-3
query G-3
queue G-3
queues, message 5-12

random record processing 4-9
reader
definition of G-3
execution 3-22
reader, job submission 3-22
record G-3
record format
definition of G-3
display file 4-23
logical file 4-12
multiple 4-15
physical file 4-11
record format level checking 4-5
record format specifications 4-2
record organization, retrieval 4-7
record retrieval
general description 4-8
random 4-9
record sequence 4-8

record, definition of 4-1
recovery 6-10
reference function, file 4-37
relationships, subsystem/job 3-14
relative record number retrieval 4-8
releasing jobs 3-19
removing routing entries 3-18
removing work entries 3-18
renaming objects 2-7
rerouting jobs 3-20
resource usage, controlling 3-2
response indicator 4-22
restore G-3
restore functions 6-10
restore objects 6-10
retrieval organization, records 4-8
revoking authority 6-7
rights of object use 6-8
routing data G-3, 3-9
routing entry
  definition of G-4
  general description 3-9
  modifying 3-18
routing step G-4, 3-9, 3-13
routing step operations 3-20

sample data, testing 5-6
save functions 6-10
save/restore 6-9
save/restore history information 6-10
save/restore operations
  library 2-8
  object 2-7
save/restore, using 6-10
save, definition of G-4
saving objects 6-9
screen formatting 4-20
searching libraries 2-5
security G-4, 6-1
security considerations 6-8
security functions 6-6
security officer G-4, 6-5
security operations, object 2-7
security, using 6-8
semi-attended operation 6-12
sequence of records 4-8
sequence, arrival 4-8
sequential record retrieval 4-8
service 6-14
sign-on, password 6-3
single level storage G-4, 1-6
source file G-4, 4-5
source statements
  data description specifications 4-31
  entering 5-6
sources, reader 3-22
special file attributes 4-5
specialization facilities 6-11
specifications
  data association 4-2
  record format 4-2
specifications form, data
  description 4-30
spooled file G-4, 4-5
spooled file processing 4-32
spooled output files 4-34
spooling functions 3-17
spooling input 3-22, 4-34
spooling subsystem G-4, 3-17
spooling, definition of G-4
starting a subsystem 3-18
step, routing 3-9, 3-13
storage allocation, objects 2-2
storage pool, definition G-4, 1-6
storage, single level 1-6
storage, transfer 1-6
structuring applications 5-5
subfiles 4-20
submitting jobs 3-19, 3-22
subsystem attributes G-4, 3-6
subsystem description
  contents 3-6
  definition of G-4, 3-3
  operations 3-18
  use of 3-4
subsystem/job relationships 3-14
subsystem monitor 3-14
subsystem operations 3-18
subsystem, definition of G-4, 3-4
subsystems
  batch 3-17
  controlling 3-4, 3-17
  CPF-provided 3-16
  general description 3-4
  managing 3-18
  spooling 3-17
  user-defined 3-18
support, display devices 4-20
system back-up 6-9
system concepts, overview 1-5
system level security 6-7
system library G-4, 2-5
system management
  facilities 6-1
  overview 1-4
system operation 6-12
system operation functions 6-12
system operation, controlling 1-2
system operator user profile 6-5
system security functions 6-6
system security officer 6-5
system tailoring 6-11

tailoring the system 6-11
temporary library G-4, 2-5
terminating a subsystem 3-18
test library G-4, 5-6
testing and debugging 5-6
testing environment 5-6
testing, library use 2-6
text descriptions 5-7
text of messages 4-23, 5-11
time slice 3-13
tracing variables 5-14
tracking file usage 4-37
training work station users 5-6
transferring object ownership 6-7
types of libraries 2-5
types of objects 2-1

unnamed inline data files 4-32
use of library list 2-6
user access to objects, controlling 6-1
user authorization 6-7
user-defined commands 5-15
user-defined subsystems 3-18
user identification G-4, 6-2
user interface 4-22
user interface, designing 5-2
user-level security 6-7
user profile
  adopting 6-9
  definition of G-4, 6-2
user-program communication 5-3
user security functions 6-7
user, work station 6-5
using data base files 4-18
using display device support 4-23
using externally described data 4-2
using messages and message queues 5-13
using save/restore 6-10
using security 6-8

variable messages 5-11
variables, debugging 5-14

where used, files 4-37
work entry
  autostart job entries 3-10
  definition of G-4
  general description 3-7
  job queue entry 3-8
  modifying 3-18
work entry (continued)
  work station entries 3-7
work management concepts 3-2
work management facilities 3-1
work management functions 3-16
work manangement, overview 1-3
work station G-4
work station entry G-4, 3-7
work station support 4-20
work station user
  definition of G-4
  job 3-3

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your nearest IBM branch office.

☐ If your comment does not need a reply (for example, pointing out a typing error) check this box and do not include your name and address below. If you comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your nearest IBM branch office to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

City State Zip Code

IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

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

GC21-7729-2

Cut Along Line

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE:

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

Fold and tape | Please do not staple | Fold and tape

IBM

International Business Machines Corporation

4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30055
(U.S.A. only)

International Business Machines Corporation

4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30055
(U.S.A. only)

IBM System/38 Control Program Facility Concepts Manual (File No. S38-36) Printed in U.S.A. GC21-7729-2