SC21-7730-0

File No. S38-36

IBM System/38

IBM System/38
Control Program Facility
Programmer's Guide

First Edition (January 1979)

Changes are periodically made to the information herein; changes will be reported in technical newsletters or in new editions of this publication.

Preface

This publication elaborates on the concepts presented in the *IBM System/38 Control Program Facility Concepts Manual,* GC21-7729, explains how to use commands described in the *IBM System/38 Control Program Facility Reference Manual—Control Language,* SC21-7731, and explains how to use data description specifications (DDS) described in the *IBM System/38 Control Program Facility Reference Manual—Data Description Specifications,* SC21-7806. This guide instructs the programmer in how to use the control program facility (CPF) functions.

This publication is organized into the following parts:

Part 1. Introduction, which includes a summary of the main CPF functions.

Part 2. Control Language, which includes a description of the System/38 control language.

Part 3. Objects, which includes a discussion of objects.

Part 4. Application Development, which includes control language programs, data management, message handling, command definition, documentation, source files, data areas, and testing.

Part 5. Work Management, which includes jobs, subsystems, job and output queues, and classes.

Part 6. System Management, which includes system values, security, save/restore, and service.

Each chapter contains the following:

- A summary of the concepts covered in the *CPF Concepts Manual* and additional topics not covered in the *CPF Concepts Manual.*
- Coding examples
- A list of commands associated with the major function so that the reader can determine which command he needs to use and to find in the *CPF Reference Manual—CL.*

A glossary at the back of this publication defines data processing terms introduced in this publication and other System/38 terms applicable to this publication.

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

Prerequisite Publications

The reader should have read the following publications:

- *IBM System/38 System Introduction,* GC21-7728, which contains an overview of the System/38 CPF, language, and utility functions as well as the system configurations.
- *IBM System/38 Control Program Facility Concepts Manual,* GC21-7729, which contains the basic concepts of the control program functions.

Related Publications

The following publications contain information related to this publication:

- *IBM System/38 Control Program Facility Reference Manual—Control Language,* SC21-7731, which describes commands and parameters that are used for the various CPF functions.
- *IBM System/38 Control Program Facility Reference Manual—Data Description Specifications,* SC21-7806, which describes the data description specifications that are used for describing files.
- *IBM System/38 Message Guide: CPF, RPG III, and IDU,* which contains additional information about CPF, RPG III, and IDU messages issued on System/38.
- *IBM System/38 Guide to Publications,* GC21-7726, which contains information about related publications.

Contents

Part 1. Introduction

Chapter 1. Introduction

Before you use this programmer's guide you should have read the *System/38 Control Program Facility Concepts*, SC21-7729. The *Concepts* manual presents a high-level description of the functions of CPF. This chapter serves as a reminder of these functions. Detailed information about the individual commands and the data description specifications can be found in the *IBM System/38 Control Program Facility Reference Manual—Control Language*, SC21-7731 and the *IBM System/38 Control Program Facility Reference Manual—Data Description Specifications*, SC21-7806, respectively.

THE CONTROL LANGUAGE

The System/38 control language is a consistent set of system commands. Commands are used to request that the system perform functions such as calling a program or holding jobs on an output queue. Commands can be entered either interactively at a display work station or in a batch job. In addition, groups of commands can be compiled into control language programs.

OBJECTS

Objects are the basic elements upon which commands perform operations. Files and programs are objects, as are print images and job queues. One type of object, the library, is useful in locating other types of objects. These other objects are placed in libraries. You can locate objects by name instead of by their actual addresses in storage.

DEVELOPING APPLICATIONS ON SYSTEM/38

Your detail-level application programs are usually written in a high-level language such as RPG. However, many CPF functions are used in developing applications:

- Creating control language programs to control the application
- Defining data
  - Defining the data base
  - Defining devices to the system
  - Overriding files for program execution
  - Copying files
  - Creating and using source files
- Defining messages
- Defining commands
- Documenting applications
- Testing application programs

WORK MANAGEMENT

Work management involves managing jobs on the system, allocating resources for use within jobs, and scheduling jobs for execution. Each job has a job description, which contains descriptive information such as job queue, job priority, initial library list, output queue, and job date.

Interactive jobs can be managed independently of batch jobs. This is accomplished through subsystems. A subsystem is an operating environment that defines a set of system resources and functions that can be used for a group of jobs.

SYSTEM MANAGEMENT

The following are CPF functions that you can use to tailor the system to your installation's needs.

- Observing and changing system values
- Securing objects against unauthorized use
- Saving and restoring objects
- Servicing System/38

Part 2. Control Language

Chapter 2. Control Language

The System/38 control language is a consistent set of commands through which you can communicate with the system. Commands are specialized in that they each perform one function. For example, there is a command to create a particular type of file and a command to create a particular type of program. Commands can also be used in writing a control language program. (See Chapter 4, *Control Language Programs* for more information.)

All System/38 commands follow the same set of rules for abbreviations and coding. For example, all create commands begin with the letters CRT; all change commands begin with the letters CHG. All other abbreviations are used consistently throughout the commands. Generally, abbreviations are three letters. There are exceptions such as F for file and Q for queue. Some words are not abbreviated (for example, the TEXT parameter).

For example, one System/38 command looks like this:

```
CRTLIB LIB(-------) TYPE(-----) PUBAUT(-------) TEXT(------)
```

CRTLIB is the coded form of the command's descriptive name, which, in this case, is Create Library. The coded name was formed from three letters of create, CRT, and three letters of library, LIB. LIB(-------) is a parameter. LIB is a keyword that indicates that a library name should be coded within the parentheses. The library name is a parameter value. Commands consist of two parts: a command name and parameters. The command name identifies the function to the system and the parameters provide additional information to tailor the function request. Some commands have no parameters. For example, the End Program command looks like this: ENDPGM

In addition, a command can be labeled by placing a label directly in front of the command name:

$$\underbrace{\text{START:}}_{\text{Label}} \quad \underbrace{\text{SNDRCVF}}_{\substack{\text{Command}\\\text{Name}}} \quad \underbrace{\text{RCDFMT(------)}}_{\text{Parameter}}$$

Labels are useful for branching within control language programs and for adding breakpoints and traces to programs. (See Chapter 15, *Testing* for more information about breakpoints and traces.)

CODING COMMANDS

Labels and names follow the same naming conventions. (See *How to Specify Names* in this chapter for more information.) Both can contain as many as 10 characters. A label is followed by a colon (no blanks can appear between the label and the colon). A command name is usually separated from a label, if there is a label, by at least one blank. However, the blank is optional. Both of the following are correct:

```
START: SNDRCVF

START:SNDRCVF
```

A command name and the first parameter in the command must be separated by at least one blank.

Parameters can be specified in two ways:

- As a *keyword parameter,* that is, a keyword with a parameter value:

  ```
  LIB(DSTPRODLB) TYPE(*PROD)
  ```

- As a *positional parameter,* that is, just a parameter value with no keyword:

  ```
  DSTPRODLB *PROD
  ```

Keyword parameters can be specified in the command in any order:

```
CRTLIB TYPE(*PROD) LIB(DSTPRODLB) PUBAUT(*NORMAL)
   TEXT('Distribution library')

CRTLIB LIB(DSTPRODLB) TYPE(*PROD) TEXT('Distribution library')
   PUBAUT(*NORMAL)
```

Positional parameters must be specified in the order in which they are described in the *CPF Reference Manual—CL*:

```
CRTLIB DSTPRODLB *PROD *NORMAL 'Distribution library'
```

If you specify parameters positionally and you want to omit a parameter, specify *N in the position of the parameter:

```
CRTLIB DSTPRODLB *N *NORMAL
```

If none of the parameters following a specified parameter are specified, *N does not have to be specified. For example, in the preceding command, if the parameter value *NORMAL is omitted, the command looks like this:

```
CRTLIB DSTPRODLB
```

You can mix positional and keyword parameters in a command. If you do so, once you use a keyword parameter all following parameters in the same command must be keyword parameters:

```
CRTLIB DSTPRODLB PUBAUT(*NONE) TEXT('Distribution library')
```

Some parameters can have more than one value. This is called a list. For example, an ALLOW parameter might look like this:

```
ALLOW(*BATCH *INTERACT)
```

The parameter values must be separated by at least one blank; no other separators are allowed. If you specify such a parameter positionally, you must still specify the parentheses:

```
(*BATCH *INTERACT)
```

The parentheses indicate that the parameter has more than one value.

For some parameters you can also specify a list within a list. For example,

```
STMT((START RESPND) (ADDDSP CONFRM))
```

Each parameter value and each list must be separated from the others by at least one blank. If you specify such a parameter in the positional form, you must still specify all parentheses:

```
((START RESPND) (ADDDSP CONFRM))
```

No matter what parameter form you use, you must leave one or more blanks between all parameters.

Continuation

If you cannot get a complete command on one line or record, you can use command continuation. There are two command continuation characters: + (plus) and - (minus). A continuation character must be the last nonblank character in a record.

A + indicates that the command continues with the first nonblank character on the next line. For example,

```
CRTLIB LIB(DSTPRODLB) TEXT('Dist+
    ribution library')
```

is equivalent to

```
CRTLIB LIB(DSTPRODLB) TEXT('Distribution library')
```

If a blank had preceded the +, the word Distribution would have looked like this:

```
Dist ribution
```

A - indicates that the command continues with the first character on the next line. For example,

```
CRTLIB LIB(DSTPRODLB) TEXT('Dist-
ribution library')
```

is equivalent to

```
CRTLIB LIB(DSTPRODLB) TEXT('Distribution library')
```

A blank before the - or at the beginning of the next line would have made the word Distribution look like this:

```
Dist ribution
```

Comments

Comments must be preceded by the symbols /* and followed by the symbols */. For example, /*Create library for distribution files*/. Comments can appear as separate records or as part of a command. Within a command comments can appear anywhere a blank is used except in a quoted string. For example, comments can appear in the following places:

- As a separate record

  ```
  /*Create library for distribution files*/
  ```

- After the last parameter

  ```
  If (&RESP='02') CALL ITM210 /*Item inquiry*/
  ```

- Between a command name and a parameter

  ```
  CRTLIB /*Create library for distribution files*/ LIB(DSTPRODLB)
  ```

- Between parameters

  ```
  LIB(DSTPRODLB) /*Production library*/ TEXT('Distribution library')
  ```

- Between parameter values

  ```
  ALLOW(*BATCH /*Batch and interactive*/ *INTERACT)
  ```

To avoid misplacing continuation characters because of comments, you should enter the comments as separate records.

Command Character Set

The extended binary coded decimal interchange code (EBCDIC) character set is used in System/38 commands. All 256 EBCDIC characters can be used in comments and quoted strings. However, only the following characters can be used in commands other than comments and quoted strings.

- 26 letters (A through Z, lowercase translated to uppercase)
- 10 digits (0 through 9)
- 3 alphabetic extenders ($, #, and @)
- 20 special characters (Figure 1)

Note: Alphameric characters are A through Z, 0 through 9, and the alphabetic extenders.

Character	Symbol	Purpose
Blank	b̸	Syntactical separator
Comma	,	Decimal point for countries using the comma as a decimal point
Equal	=	Relational equal operator
Plus	+	Addition operator, continuation character, and prefix plus
Minus	-	Subtraction operator, continuation character, and prefix minus
Asterisk	*	Multiplication operator, suffix for generic name, prefix for predefined value, and part of comment symbols (/* */)
Slash	/	Division operator and part of comment symbols (/* */)
Left parenthesis	(	Delimiter for parameter values and lists, and defines order of expressions
Right parenthesis	)	Delimiter for parameter values and lists, and defines order of expressions
Apostrophe	'	Character literal delimiter
Not	¬	Logical negation operator
Or	|	Logical OR operator and concatentation operator when paired ||
Less than	<	Relational less than operator
Greater than	>	Relational greater than operator
Period	.	Decimal point and connector for qualified name
Ampersand	&	Logical AND operator and prefix for symbolic variable
Percent	%	Prefix for built-in function
Colon	:	Delimiter for command labels
Underscore	_	Standard alphabetic character
Question mark	?	Prompt request

Figure 1. Special Characters

Types of Parameter Values and How to Specify Them

Four types of values can be specified for parameters:

- Constant
  - Character
  - Decimal
  - Logical
  - Hexadecimal

- Variable
  - Decimal
  - Character
  - Logical

- Expression
  - Built-in function
  - Arithmetic
  - Logical
  - Character
  - Relational

For information about variables and expressions, see Chapter 4, *Control Language Programs*.

The following sections explain how to specify parameters using constant data. The types of constants discussed are:

- Character
  - Character string
  - Name
  - Date
  - Time

- Decimal

- Logical

How to Specify Character Strings

There are two types of character strings: quoted and unquoted. The difference between the two types is determined by the use of apostrophes and characters allowed in the string. A quoted string contains any EBCDIC characters enclosed in apostrophes:

```
'Bob Smith–Accounting dept 512'
```

Note: When the apostrophe (') is used within a quoted string, it must be specified as two single apostrophes (''):

```
'Bob''s accounting files'
```

An unquoted string can contain only alphameric characters, a leading plus (+) or minus (-), and one period (.) or comma (,) as a decimal point:

```
+2.1
FILEA
```

An unquoted string can consist of all numeric characters or all alphameric characters. A numeric unquoted string can be preceded by a + or - sign, and contain a decimal point (. or ,). An unquoted string with at least one alphabetic character cannot contain a plus, minus, or comma. An alphameric unquoted string can contain a name or an * (asterisk) followed by a name. A period can only be used for a qualified name.

The following table shows which characters can be used in which character strings. (X indicates what is valid.)

Character	Decimal String or Numeric Value	Unquoted String	Quoted String
Letters			X
A-Z		X	X
a-z		X[1]	X
Alphabetic extenders			
&		X	X
@		X	X
#		X	X
!			X
"			X
¢			X
Underscore _		X	X
Digits 0-9	X	X	X
Comma ,	X[2]		X
Period .	X[2]		X
Blank ƀ			X
Parentheses ( )			X
Apostrophe '			X[3]
Colon :			X
Percent %			X
Question mark ?			X
Relational operators			
=			X
<			X
>			X
Logical operators			
|			X
&			X
¬			X
Arithmetic operators			
+[4]	X		X
-[4]	X		X
/			X
*		X[5]	X

Notes:
[1]Translated to uppercase letters.
[2]Only one allowed.
[3]Must be specified as two apostrophes but is still considered one character.
[4]Except on the CALL command and in an expression, a numeric unquoted string can have a leading plus or minus sign.
[5]Can only be the first or last character; only one allowed.

How to Specify Names

A name is a special form of an unquoted character string. Names are used to identify objects such as files and programs. Names are restricted to a maximum of ten characters. The first character must be alphabetic and the remaining characters alphameric:

```
ORD040C
ORDHDRP
```

If a name is followed by an asterisk (*), the name is generic. A *generic name* identifies a group of objects. A generic name consists of one or more letters that are common to the beginnings of a group of object names, and is followed by an *. For example, ORD* identifies ORDHDRP, ORD040C, and ORDFILL. For example, ORD* might be specified in the Display Object Description (DSPOBJD) command to display the descriptions of more than one object at a time.

Another form of a name is a *qualified name*. A qualified name is an object name, followed by a period, followed by a library name:

```
ORDFILL.DSTPRODLB
```

The library name qualifies the object name by indicating in what library the referenced object is located. This helps you get the exact object you want. (See Chapter 3, *Objects* for more information.)

How to Specify Date and Time

Date and time are both special forms of an unquoted or quoted character string. When date and time are specified as unquoted strings, they cannot contain special characters to delimit the parts of the value. When date and time are specified as quoted strings, they can contain special characters.

When date is specified as an unquoted string, the date contains six digits. When date is specified as a quoted string, the date contains six digits with day, month, and year separated by date separators. The separator is specified in the system value QDATSEP. For a quoted date string, leading zeros for day and month can be dropped. For example, when the date format is month, day, and year in that order, a date can look like this:

Unquoted	**Quoted**
060380	'06/03/80' or '6-03-80' or '6/3/80'

Date can be specified in any one of the following formats, whichever is specified as the format for your system (QDATFMT system format):

- Year, month, day
- Month, day, year
- Day, month, year
- Julian (year, day)

(See Chapter 17, *System Values* for how the date format is specified.)

Time of day is always specified in this order: hours, minutes, and, optionally, seconds. When time is specified as an unquoted string, the time contains no separators. When time is specified as a quoted string, colons are used as separators and leading zeros for hours can be dropped.

Unquoted	**Quoted**
100215	'10:02:15'
0806	'8:06'

Note: The system date and time are contained in the system values QDATE and QTIME.

How to Specify Decimal Values

Decimal constants specify parameter values such as block length and priority level. A decimal constant can contain from one through 15 digits optionally preceded by a plus (+) or minus (-). In addition, a decimal constant can contain a decimal point (. or ,), but no more than nine digits can be to the right of the decimal point. Examples are:

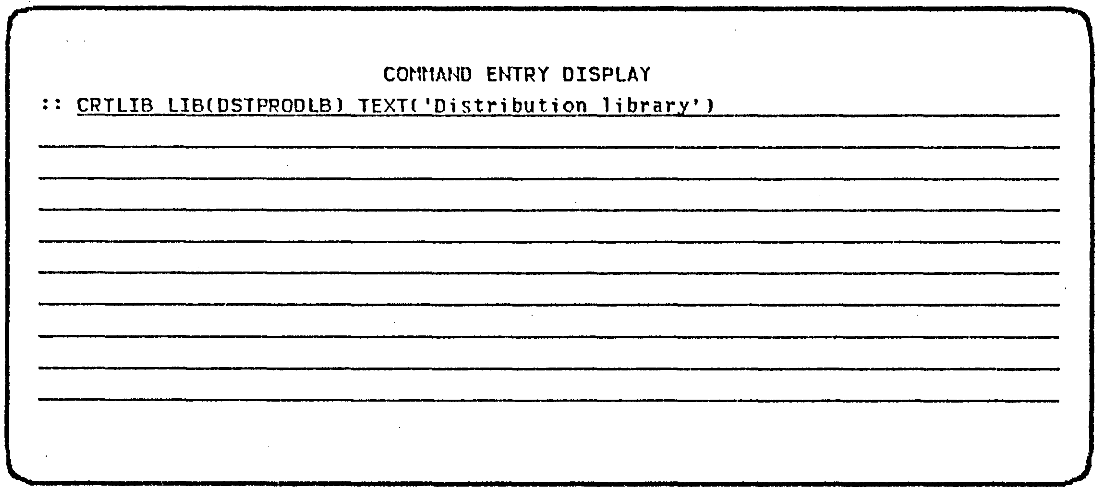
```
123
-123
12.3
-.123
```

How to Specify Logical Values

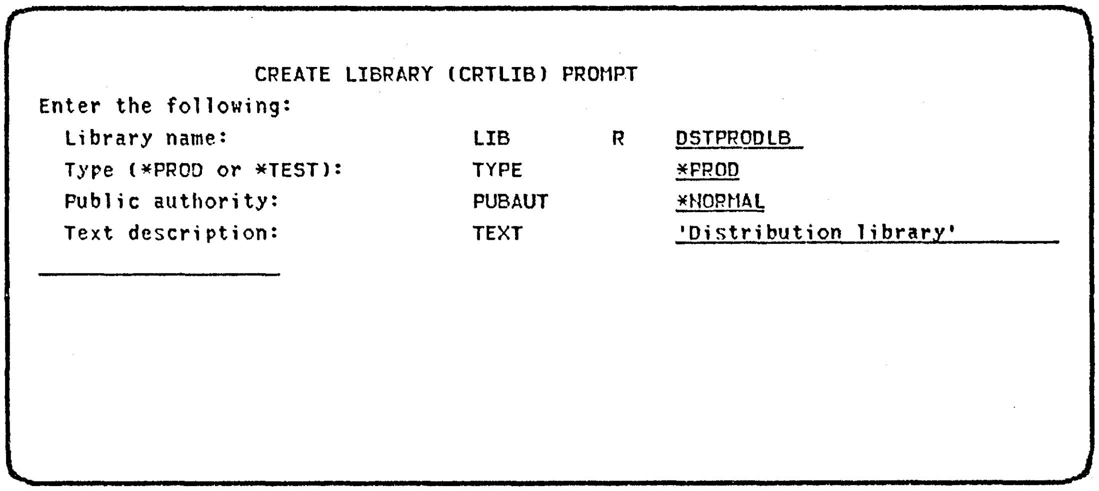
Logical constants can only be '0' or '1'. A logical 0 or 1 must always be enclosed in apostrophes.

SUBMITTING COMMANDS

Commands can be submitted to the system through two environments: interactive and batch. Interactive commands are entered at a work station on the command entry display. You can enter an entire command or a partial command. By entering a partial command, you request that the system help you. You can request menus from which you can select command names. In addition, you can request that the system prompt you for parameter values. Another way of executing commands interactively is to compile a group of commands into a control language program and then to use the CALL command to execute the program. The following displays show a partial command CRTLIB entered on the command entry display and the prompt for the same command.



```
                          COMMAND ENTRY DISPLAY
:: CRTLIB LIB(DSTPRODLB) TEXT('Distribution library')
```



```
              CREATE LIBRARY (CRTLIB) PROMPT
Enter the following:
  Library name:                   LIB      R   DSTPRODLB
  Type (*PROD or *TEST):          TYPE         *PROD
  Public authority:               PUBAUT       *NORMAL
  Text description:               TEXT         'Distribution library'
```

See the *Work Station User's Guide* for a complete discussion of entering commands interactively.

In the batch environment, commands are entered from card readers, diskettes, or data base source files. These commands can be interpretively executed or compiled into a control language program that is executed by using the CALL command.

Part 3. Objects

Chapter 3. Objects

Objects are the basic units upon which commands perform operations. For example, programs and files are objects. Objects are placed in libraries, which are themselves objects. You can use objects by identifying them by name, and optionally by a library name. You do not know the storage address of an object.

The valid object types for System/38 are:

- Class
- Command definition
- Control unit description
- Data area
- Device description
- Edit description
- File
- Job description
- Job queue
- Library
- Line description
- Message file
- Message queue
- Output queue
- Print image
- Program
- Subsystem description
- Table
- User profile

Each of these types of objects has a set of attributes that describes the object. The common attributes are:

- Name. The name given to the object when it was created.
- Type. The type of object it is.
- Owner name. The name of the user who owns the object; that is, the user who created the object or who received ownership of the object. (Only one user at a time can own an object.)
- Creation date. The date on which the object was created.
- Save date. The date on which the object was last saved.
- Volume identifier for save. The volume on which the object was last saved.
- Free storage indicator. Whether the object's storage was freed when the object was saved.
- Text description. Up to 50 characters that describe the object. The user specifies the text when he creates the object.

The following is a list of functions that apply to objects in general. There are also specific functions such as create, change, and display for each object type. The specific functions are described in other sections of this publication that describe the object type (for example, Chapter 4, *Control Language Programs* describes creating programs).

- Renaming objects
- Moving objects between libraries
- Displaying object descriptions
- Allocating and deallocating objects (see Chapter 16, *Work Management*)
- Changing ownership of an object (see Chapter 18, *Security*)
- Granting and revoking authority for objects (see Chapter 18, *Security*)
- Saving and restoring objects (see Chapter 19, *Save/Restore*)
- Dumping objects (see Chapter 15, *Testing*)

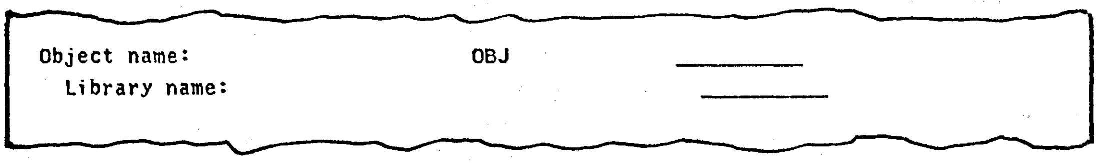
All objects are identified by names and types. Sometimes the object type is implied in the command; the type is not specified. For example, the CRTLIB (Create Library) command implies that the object type is library. Sometimes the command does not imply the object type, and the object type can be specified in the OBJTYPE parameter. For example, the DSPOBJD (Display Object Description) command applies to any type of object. In addition, objects can be identified by their libraries. All objects except libraries, device descriptions, line descriptions, control unit descriptions, and user profiles are in libraries. By specifying a library name with an object name you provide a unique name for each object on the system. Object names except libraries can have two parts: an object name followed by a library name. When both parts are specified, the name is said to be a qualified name. For example, if you are entering a command in which you must specify an object name, the object name could be:

ORD040C.DSTPRODLB

The order entry program ORD040C is in the library DSTPRODLB.

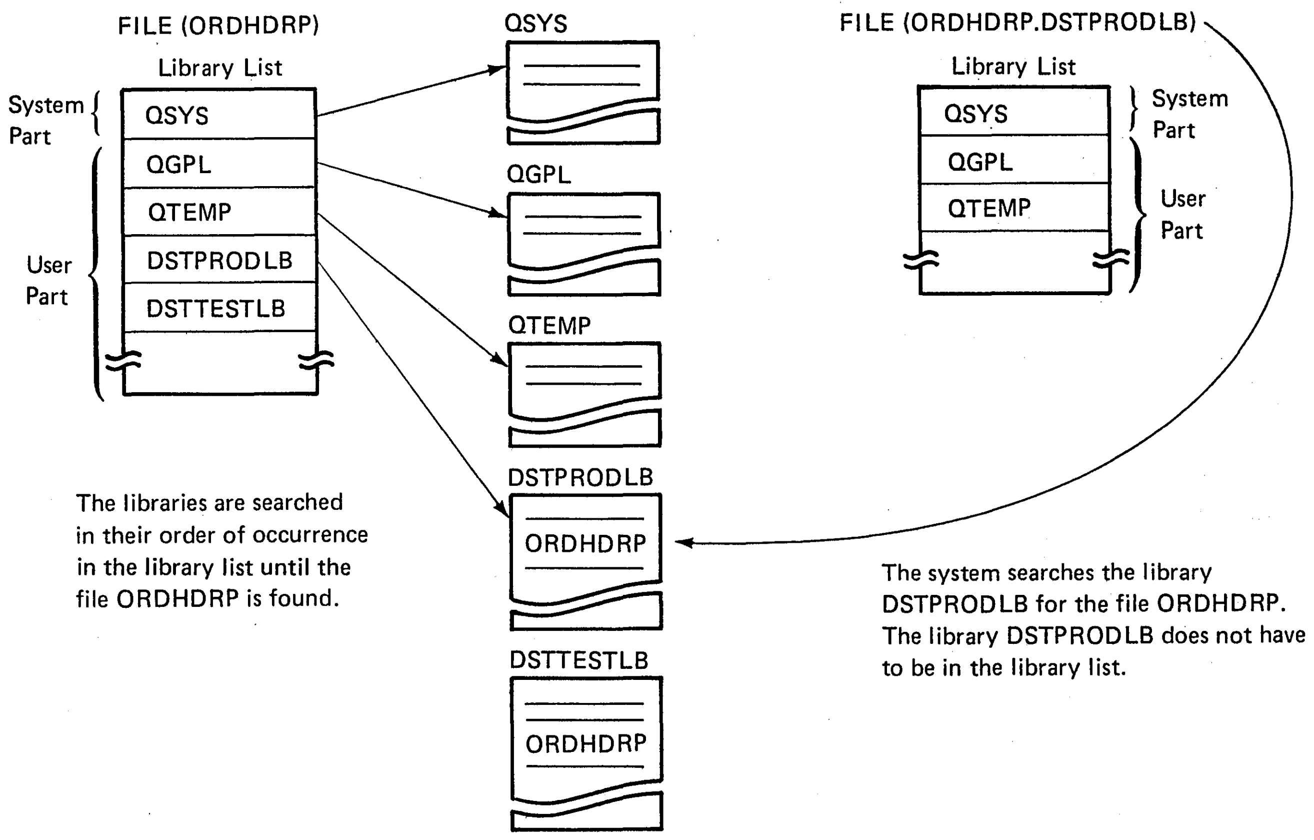
Note: When you specify an edit description other than when you create it, you cannot specify a library name.

If you are using prompting during command entry, you receive two prompts, one for object name and one for library name:



```
Object name:                    OBJ         __________
  Library name:                               __________
```

In most commands you can specify an object name without specifying a library name. The system searches the job's list of libraries, the library list, to find the object. However, if two or more objects of the same type exist in the library list with the same name, you get the first object (for which you have authority) with the specified name and type encountered in the list. If you want to make sure you get a particular object, you can either (1) specify the correct library when you specify the object name or (2) place the library with the object you want in the library list before other libraries containing an object with the same name and type. The following shows the searches made for an object both when no library name is specified and when a library name is specified.



The libraries are searched in their order of occurrence in the library list until the file ORDHDRP is found.

The system searches the library DSTPRODLB for the file ORDHDRP. The library DSTPRODLB does not have to be in the library list.

A library list consists of two parts: a system part and a user part. Initially, the system part of the library list contains QSYS and the user part contains QGPL and QTEMP. These can be changed in two ways:

- For all jobs you can change the initial library list by changing the system values QSYSLIBL and QUSRLIBL. These system values control what the initial library list is for all jobs on the system. Use the Change System Value (CHGSYSVAL) command to change either value. QSYSLIBL can contain five libraries; QUSRLIBL can contain 10 libraries.

- For individual jobs you can change, after the job has started, the user part of the library list, but not the system part. You can use any of the following commands to change the user part of the library list.
  - Replace Library List (RPLLIBL) command
  - Create Job Description (CRTJOBD) command
  - Job (JOB) command

For example, you want to use objects in DSTPRODLB, but you do not want to specify the library name every time you reference an object in the library. The initial library list for a job is always (if QSYSLIBL and QUSRLIBL have not changed and the job description for a job does not specify a different initial library list):

1. System library (QSYS)

2. General purpose library (QGPL)

3. Temporary library (QTEMP)

By issuing an RPLLIBL command you can change the user part of the library list for your job:

```
RPLLIBL LIBL(DSTPRODLB QGPL)
```

The library list for your job becomes:

1. QSYS (system part)

2. DSTPRODLB (user part)

3. QGPL (user part)

This change affects only one job; it is not permanent; it exists only as long as the job is executing. The initial library list is (1) QSYS, (2) QGPL, and (3) QTEMP again for each new job (or whatever is specified in QSYSLIBL and QUSRLIBL).

SECURITY CONSIDERATIONS

When you create objects you can specify through the PUBAUT parameter the public authority for an object. Public authority is the authority granted to all users. Public authority can be specified as:

- Normal (all users have some authority to use the object)
- None (only the owner can use the object)
- All (every user can use an object as if he were the owner)

Also, an owner can grant authority for his object to specific users. Chapter 18, *Security* explains in detail the types of authority for an object.

LIBRARIES

Libraries let you group objects according to application, user, department, or anything you want. For example, you might place all your order entry files and programs into an order entry library DSTPRODLB, or you might place all the files that a user JOE can use in a library JOELIB. If you placed all your order entry files and programs in DSTPRODLB, you need only add one library to the library list to ensure that all your order entry files and programs are in the list. This is advantageous if you do not want to specify a library name every time you use an order entry file or program. Also, it is advantageous when you are testing (see Chapter 15, *Testing*).

What you want to put in a library determines which type of library it is. There are two types of libraries: production and test. A production library is for normal processing. A test library is used in debug mode. In debug mode, you can either update:

- Any data base file, whether in a test or production library, or
- Only data base files in a test library. You can use files and programs in a production library, but you cannot update them. (See Chapter 15, *Testing* for more information.)

In addition, multiple libraries make it easier to use objects. For example, you can have two files with the same name but in different libraries so that one can be used for testing and the other for normal processing. As long as you do not specify the library name in your program the file name in the program does not have to be changed for testing or normal processing.

Note: Objects of the same type can have the same names only if they are in different libraries.

To create a library, you use the Create Library (CRTLIB) command. The following CRTLIB command creates a library to contain all your order entry files and programs. The library is named DSTPRODLB and is a production library.

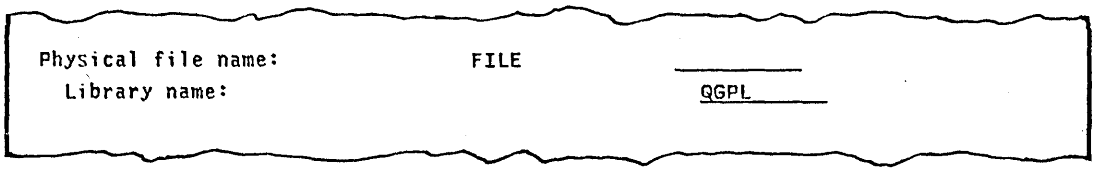
```
CRTLIB LIB(DSTPRODLB) TYPE(*PROD)
        PUBAUT(*NORMAL)  TEXT('Distribution library')
```

PLACING OBJECTS IN LIBRARIES

When you create an object (other than a library, user profile, device description, line description, or control unit description), you must place it in a library. If you do not specify a library, the object is placed in the IBM-supplied general purpose library QGPL. To specify a library you specify a qualified object name, that is, an object name and a library name. For example, you create an order entry physical file ORDHDRP, which you want to place in DSTPRODLB. In the Create Physical File (CRTPF) command, you specify in the name of the file as:

```
FILE(ORDHDRP.DSTPRODLB)
```

Note: If you are using prompting instead of command entry, you receive two prompts, one for object name and one for library name.



```
Physical file name:              FILE        __________
  Library name:                                QGPL______
```

DISPLAYING OBJECT DESCRIPTIONS

You can display descriptions of objects. If you are using batch processing, the descriptions are printed. If you are using interactive processing, the descriptions can be displayed or printed.

You can display basic or full object descriptions. The basic and full descriptions for objects are:

Basic	**Full**
Object name	Object name
Library name	Object type
Object type	Creation date
Extended attributes	Object size
Online status	Online status
Text description (partial)	Owner name
	Library name
	Date saved
	Saved location
	Extended attributes
	Text description

Using the Display Object Description (DSPOBJD) command, you can list the objects in a library by:

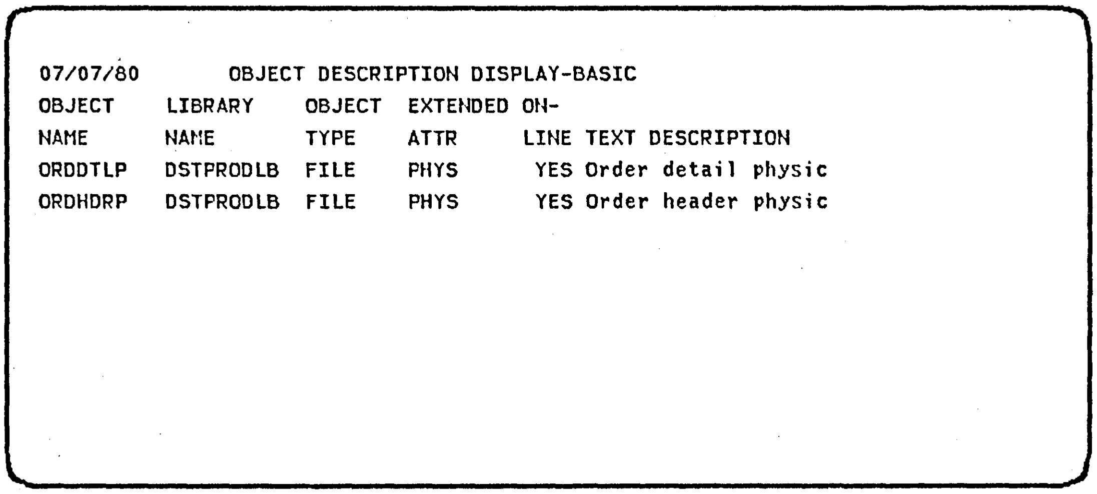
- Name
- Generic name
- Type
- Name or generic name within object type

The objects are listed according to type. Within object type the objects are listed in alphameric order.

In the following example, you display the descriptions of your order entry files (that is, the files in DSTPRODLB) whose names begin with ORD. ORD* is the generic name.

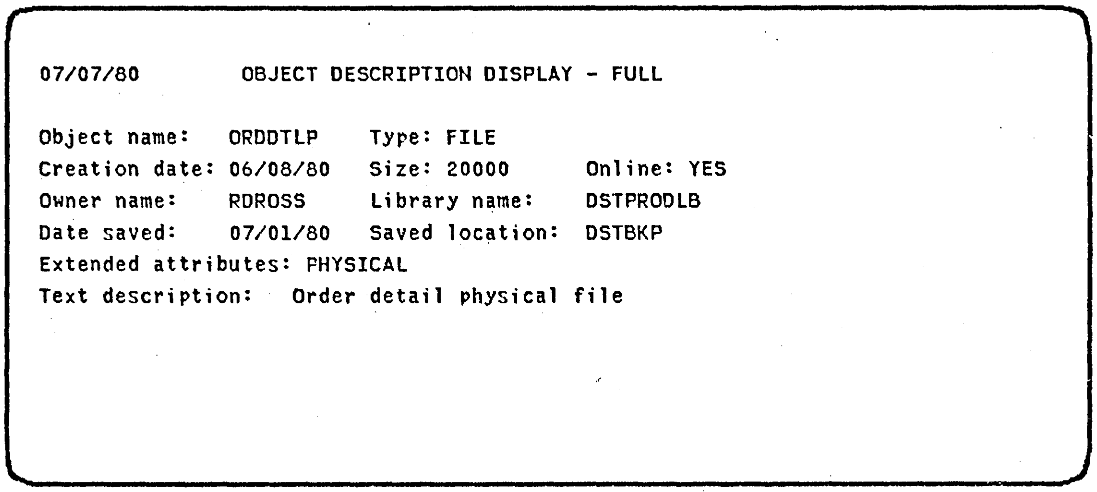
```
DSPOBJD OBJ(ORD*.DSTPRODLB) OBJTYPE(FILE)
     DETAIL(*BASIC) OUTPUT(*)
```

The resulting basic display is:



```
07/07/80      OBJECT DESCRIPTION DISPLAY-BASIC
OBJECT    LIBRARY    OBJECT  EXTENDED ON-
NAME      NAME       TYPE    ATTR     LINE TEXT DESCRIPTION
ORDDTLP   DSTPRODLB  FILE    PHYS      YES Order detail physic
ORDHDRP   DSTPRODLB  FILE    PHYS      YES Order header physic
```

If you specify *FULL instead of *BASIC, the resulting full display is:

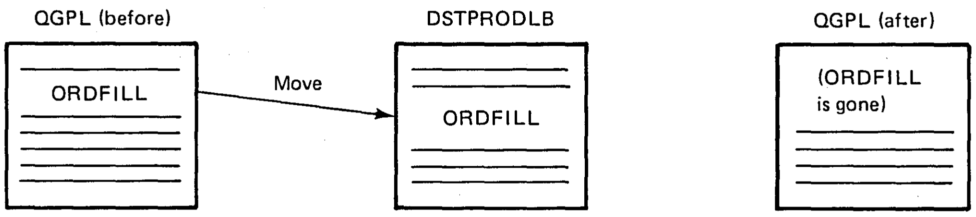


```
07/07/80       OBJECT DESCRIPTION DISPLAY - FULL

Object name:   ORDDTLP    Type: FILE
Creation date: 06/08/80   Size: 20000      Online: YES
Owner name:    RDROSS     Library name:    DSTPRODLB
Date saved:    07/01/80   Saved location:  DSTBKP
Extended attributes: PHYSICAL
Text description:   Order detail physical file
```

MOVING OBJECTS FROM ONE LIBRARY TO ANOTHER

All objects except libraries, user profiles, line descriptions, control unit descriptions, device descriptions, and edit descriptions can be moved between libraries. However, you can only move an object if you have object management rights for the object, delete rights for the library the object is being moved from, and add rights to the library the object is being moved to.

In the following example, you move a file from QGPL (where it was placed by default when you created it) to your order entry library DSTPRODLB so that it is grouped with your other order entry files.



To move the object you must specify the to library as well as the object type.

```
MOVOBJ OBJ(ORDFILL.QGPL)  OBJTYPE(FILE) TOLIB(DSTPRODLB)
```

RENAMING OBJECTS

All objects except QTEMP, user profiles, line descriptions, control unit descriptions, device descriptions, and edit descriptions can be renamed. However, you can only rename an object if you have object management rights for the object and update rights for the library containing the object.

To rename an object you must specify the current name of the object, the name to which the object is to be renamed, and the object type.

You rename an object using the Rename Object (RNMOBJ) command. The following RNMOBJ command renames the object ORDERL to ORDFILL.

```
RNMOBJ  OBJ(ORDERL.QGPL) OBJTYPE(FILE)  NEWOBJ(ORDFILL)
```

Note that you do not specify a qualified name for the new object name because the object remains in the same library.

ALLOCATING RESOURCES

Generally, objects are allocated on demand; that is, when a routing step needs an object it obtains (locks) the object, uses the object, and deallocates (unlocks) the object so another routing step can use it. Sometimes you want to allocate an object for a job before the job that is processing needs the object. This is called preallocating an object. You would preallocate an object so that you would be ensured of the availability of the object. A function that has only partially completed would not have to wait for an object.

Objects are allocated on the basis of their intended use (read or update) and whether they can be shared (used by more than one job). A lock state identifies the use of the object and whether it is shared. The five lock states are:

- Exclusive. The object is reserved for the exclusive use of the requesting routing step; no other routing steps can use the object. However, if the object is already allocated to another routing step, your routing step cannot get exclusive use of the object.
- Exclusive allow read. The object is allocated to the routing step that requested it, but other routing steps can read the object if they request a shared for read lock state or a shared no update lock state for the same object.
- Shared for update. The object can be shared either for update or read with another routing step. That is, another routing step can request either a shared for read lock state or a shared for update lock state for the same object.

- Shared no update. The object can be shared with another routing step if the routing step requests either a shared no update lock state, a shared for read lock state, or an exclusive allow read lock state.

- Shared for read. The object can be shared with another routing entry if the routing entry does not request exclusive use of the object. That is, another routing step can request an exclusive allow read, shared for update, shared for read, or shared no update lock state, or an exclusive allow read lock state.

The following shows the valid lock state combinations for an object:

If One Routing Step Obtains This Lock State:	Another Routing Step Can Obtain This Lock State:				
	Exclusive	Exclusive Allow Read	Shared for Update	Shared No Update	Shared for Read
Exclusive					
Exclusive Allow Read					X
Shared for Update			X		X
Shared No Update				X	X
Shared for Read		X	X	X	X

To allocate an object use the Allocate Object (ALCOBJ) command. To deallocate an object use the Deallocate Object (DLCOBJ) command. Allocated objects are automatically deallocated at the end of a routing step.

The following example is a batch job that needs two files for updating. Members from either file can be read by another program while being updated, but no other programs can update these members while this job is executing. The first member of each file is preallocated with a shared no update lock state.

```
//JOB JOBD(ORDER)
 ALCOBJ OBJ((FILEA FILE *SHRNUP) (FILEB FILE *SHRNUP))
 CALL PROGX
//ENDJOB
```

Deallocate Object (DLCOBJ) commands do not have to be specified; the objects are automatically deallocated at the end of the routing step.

If the first members of FILEA and FILEB had not been preallocated, the shared no update restriction would not have been in effect.

DELETING AND CLEARING LIBRARIES

When you delete a library, you delete the objects in the library, the library description, and the library itself. When you clear a library, you delete objects in the library without deleting the library. To delete or clear a library all you need specify is the library name (for example, DLTLIB LIB(DSTPRODLB)).

To delete a library you must have object existence rights for both the library and the objects within the library. If you try to delete a library but do not have object existence rights for all the objects in the library, the library and all objects for which you do not have authority are not deleted. All objects for which you have authority are deleted. If you want to delete a specific object (for which you have object existence rights), you can use a delete command for that type of object (such as the Delete Program command).

To clear a library you must have object existence rights for the objects within the library. If you try to clear a library but do not have object existence rights for all the objects in the library, the objects you do not have rights for are not deleted from the library.

DISPLAYING LIBRARIES

You can display the objects contained in one or more libraries. All authorized objects' names and types are listed. If you are using batch processing, the list is printed. If you are using interactive processing, the list can be displayed or printed.

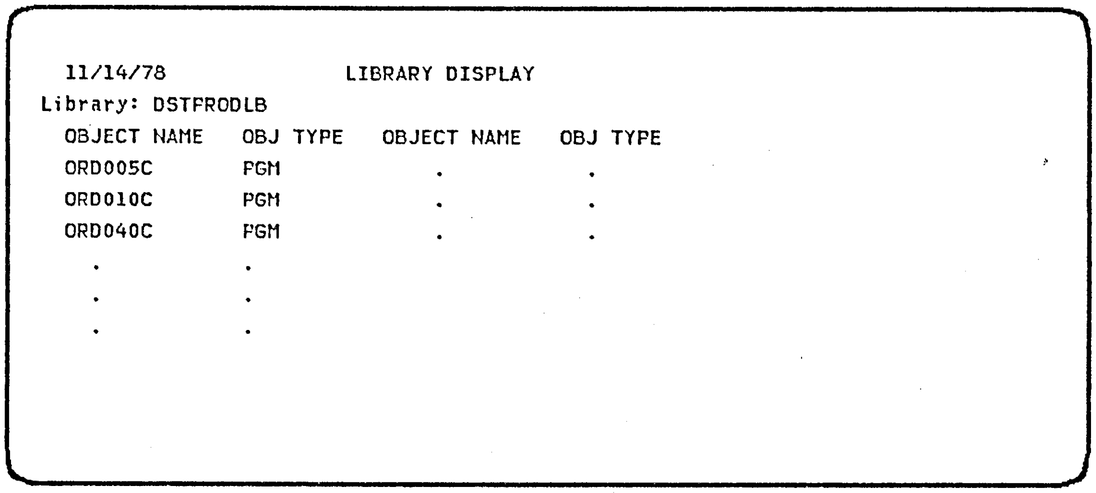
In the list, the objects are grouped by library; within each library, they are grouped by object type; within each type, they are listed in alphameric order. The order of the libraries matches the library list (*LIBL) or the order of the libraries specified in the display command.

For example, the following Display Library (DSPLIB) command displays a list of the objects contained in DSTPRODLB.

```
DSPLIB LIB(DSTPRODLB) OUTPUT(*)
```

The * (asterisk) for the OUTPUT parameter means that the list is to be displayed if in interactive processing and printed if in batch processing. (To print a list when in interactive processing, specify *LIST instead of taking the default *.)

The resulting display is:

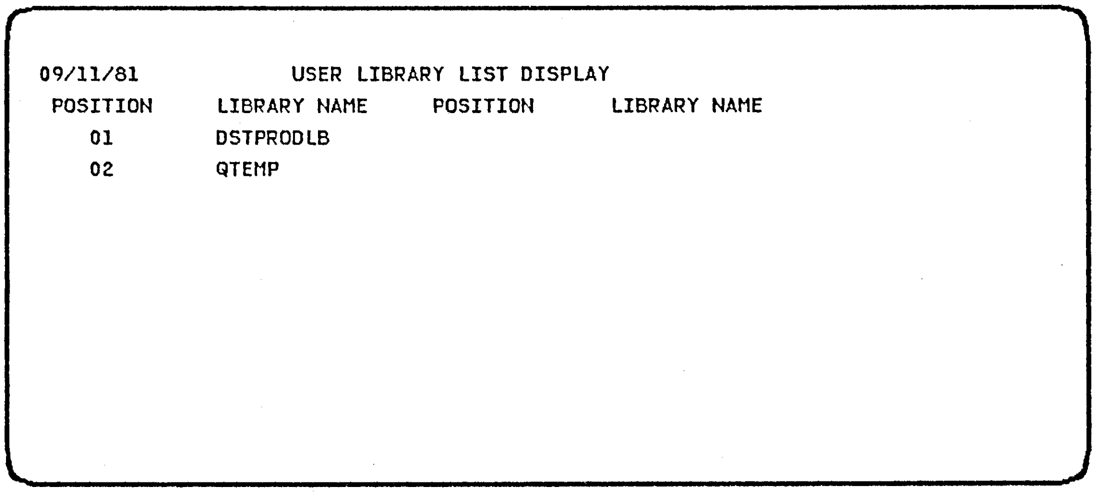


```
  11/14/78              LIBRARY DISPLAY
Library: DSTFRODLB
  OBJECT NAME   OBJ TYPE   OBJECT NAME   OBJ TYPE
  ORD005C       PGM             .            .
  ORD010C       PGM             .            .
  ORD040C       PGM             .            .
     .             .
     .             .
     .             .
```

DISPLAYING A LIBRARY LIST

You can display the user part of the library list for a job currently executing. The display is a list of all the libraries in the library list in the order in which they appear in the library list. If you are using batch processing, the library list is printed. If you are using interactive processing, the library list can be displayed or printed.

You use the Display Library List (DSPLIBL) command to show the list. The following is an example of a display of a library list.



```
09/11/81          USER LIBRARY LIST DISPLAY
 POSITION     LIBRARY NAME     POSITION      LIBRARY NAME
    01        DSTPRODLB
    02        QTEMP
```

COMMAND LIST

This is a list of commands related to objects in general. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL.*

Objects

Descriptive Name	Command Name	Function
Rename Object	RNMOBJ	Changes the name of an object. However, device descriptions, control unit descriptions, line descriptions, edit descriptions, and user profiles cannot be renamed.
Move Object	MOVOBJ	Moves an object from one library to another.
Display Object Description	DSPOBJD	Displays descriptions of specified objects.
Allocate Object	ALCOBJ	Reserves an object for use by a job.
Deallocate Object	DLCOBJ	Frees an object from reserved use by a job.

Libraries

Descriptive Name	Command Name	Function
Create Library	CRTLIB	Creates a library.
Delete Library	DLTLIB	Deletes a library and all objects in the library. However, the user must have object existence rights to the library and all its objects.
Clear Library	CLRLIB	Deletes all objects to which the user has object existence rights from a specified library. However, the library is not deleted.
Display Library	DSPLIB	Displays a list of the names and object types of all objects in a specified library.
Replace Library List	RPLLIBL	Replaces the user part of the library list with a new set of library names.
Display Library List	DSPLIBL	Displays a list of all libraries in the user part of the library list.

Other Commands

This is a list of commands that are also related to objects in general but are not part of the functions presented in this chapter.

Descriptive Name	Command Name	Chapter
Change Object Owner	CHGOBJOWN	Chapter 18, Security
Grant Object Authority	GRTOBJAUT	Chapter 18, Security
Revoke Object Authority	RVKOBJAUT	Chapter 18, Security
Display Object Authority	DSPOBJAUT	Chapter 18, Security
Save Object	SAVOBJ	Chapter 19, Save/Restore
Restore Object	RSTOBJ	Chapter 19, Save/Restore
Dump Object	DMPOBJ	Chapter 15, Testing
Save Library	SAVLIB	Chapter 19, Save/Restore
Restore Library	RSTLIB	Chapter 19, Save/Restore

Part 4. Application Development

Chapter 4. Control Language Programs

You can group together most commands and compile them so that they can be invoked by a single command. The compiled commands are called a *control language program*. You can perform functions with a control language program that you cannot perform with simple command entry. Some of these functions are:

- Manipulating variables and data areas
- Executing commands conditionally
- Grouping commands for conditional execution
- Using a built-in function
- Monitoring for messages
- Sending and receiving information to and from a device
- Passing parameters to called programs

CONTROL LANGUAGE PROGRAM VARIABLES

A *variable* is a named changeable value that can be accessed or changed by referring to its name. Variables can be used as substitutes for parameter values on control language commands. When a CL program variable is used as a parameter value and the command containing it is executed, a value is substituted for the variable. Every time the command is executed a different value can be substituted for the variable.

CL program variables can exist only within control language programs. Program variables are not stored in libraries; they are not objects; and their values are destroyed when the program that contains them is no longer invoked.

CL program variables are defined by Declare (DCL) commands, a Declare Data Area (DCLDTAARA) commands, or Declare File (DCLF) commands.

When a DCLF command is used, the control language compiler declares CL variables for each field and option indicator in each device file record format. For a field, the CL variable name is the field name preceded by an ampersand (&). For an option indicator, the CL variable name is the indicator preceeded by &IN.

When a DCLDTAARA command is used, a variable is defined with the same attributes as the data area. The variable name is the same as the data area but is preceded by an &.

When a DCL command is used, the following restrictions must be followed:

- The CL variable name must begin with an ampersand (&) followed by as many as 10 characters. The first character must be alphabetic and the remaining characters alphameric. For example, &PART.

- The CL variable value must be either character, decimal, or logical.
  - A character string as long as 2000 characters
  - A decimal value as long as 15 digits with as many as nine decimal positions
  - A logical value '0' or '1', where '0' can mean off, false, or no; and '1' can mean on, true, or yes

- If you do not specify an initial value, the following is assumed:
  - 0 for decimal
  - Blanks for character
  - '0' for logical

For example, variables can be used when you create objects. A variable can be used in place of the object name or the library name or both:

```
CRTPF  FILE(&FILE.DSTPRODLB) . . .
CRTPF  FILE(ORD040C.&LIB) . . .
CRTPF  FILE(&PFILE.&LIB) . . .
```

You can change the value of a CL program variable using the Change Variable (CHGVAR) command. The value can be changed:

- To a constant:

  ```
  CHGVAR  VAR(&INVCMPLT)  VALUE(0)
  ```

  &INVCMPLT is set to 0.

- To the value of another variable:

  ```
  CHGVAR  VAR(&A)  VALUE(&B)
  ```

  &A is set to the value of the variable &B.

- To the value of an expression after it is evaluated:

  ```
  CHGVAR  VAR(&A)  VALUE(&A + 1)
  ```

  The value of &A is increased by one.

- To the value produced by the built-in function %SUBSTRING:

  ```
  CHGVAR  VAR(&A)  VALUE(%SUBSTRING(&B 1 5))
  ```

  &A is set to the first five characters of the value of the variable &B. (See *Substring Built-in Function* in this chapter for more information.)

The variable and the value to which the variable is to be changed must be of the same type (character, logical, or decimal).

CONTROLLING LOGIC FLOW

Logic flow within a control language program can be controlled conditionally and unconditionally. Conditional execution is based on a logical expression and is accomplished using the IF command. If an expression is true, the command following the THEN keyword is executed. If the expression is false, the command following the THEN keyword is not executed, but the ELSE command (if specified) is executed. The THEN keyword does not have to be specified. When there is no ELSE command or when the command following the THEN keyword has executed, the next executable command is executed. The following is an example of conditional execution.

```
IF (&RESP=1) THEN(CALL CUS210)
```

If the variable &RESP equals one, (then) call the program CUS210.

Commands can be grouped for conditional execution after the THEN keyword, after the IF expression if THEN is not specified, after the ELSE command, or after you use a DO command to group the commands. The following is an example of a DO command.

```
IF (&A=YES)
  DO
    CHGVAR VAR(&B) VALUE(1)
    CHGVAR VAR(&C) VALUE('Z')
  ENDDO
```

Do groups can be nested (embedded) ten levels. A nested do group is a do group that is contained within another do group:

```
DO
•
•
•
  DO
  •
  •
  •
  ENDDO
•
•
•
ENDDO
```

Unconditional branching is accomplished using the GOTO command. To use a GOTO command, the command you are branching to must have a label:

```
      •
      •
      •
START: SNDRCVF RCDFMT(MENU)
      IF (&RESP=1) THEN(CALL CUS210)
      •
      •
      •
      GOTO START
```

SUBSTRING BUILT-IN FUNCTION

The substring built-in function (%SUBSTRING) produces a character string that is a subset of an existing character string. It can only be used in the Change Variable (CHGVAR) and IF commands, and can only be used within a control language program. In a CHGVAR command %SUBSTRING can be specified in place of the variable (VAR parameter) to be changed or the value (VALUE parameter) to which the variable is to be changed. In an IF command %SUBSTRING can be specified in the expression.

The format of the substring built-in function is:

```
%SUBSTRING(character-variable-name start length)
```

The following is an example of %SUBSTRING used in an IF command.

```
IF (%SUBSTRING(&NAME 1 2) *EQ 'IN')
   THEN(CALL INV210 &NAME)
ELSE CHGVAR &ERRCODE 99
```

If the first two positions in the variable &NAME are IN, the program INV210 is called. The entire value of &NAME is passed to INV210. Otherwise, the value of &ERRCODE is set to 99.

WRITING CONTROL LANGUAGE PROGRAMS

Control language programs can be written for many purposes, including:

- To call a series of HLL batch programs
- To display a menu and determine the programs to be called as a result of options selected from the menu
- To simplify work station and system operations by placing a commonly used sequence of commands in a control language program
- To control the sequence of processing
- To automate a series of functions for use as an initial program (see Chapter 18, *Security*) or as a program for a routing step (see Chapter 16, *Work Management*)

As you write control language programs you should keep in mind the following information.

- A control language program can contain only control language commands.

- A control language source program should begin with a Program (PGM) command and end with an End Program (ENDPGM) command. If not specified, PGM and ENDPGM are assumed. Only one PGM command can be in a source program; nested (embedded) programs are not allowed.

- All declare commands must follow the PGM command, if specified, and precede any other commands in the program.

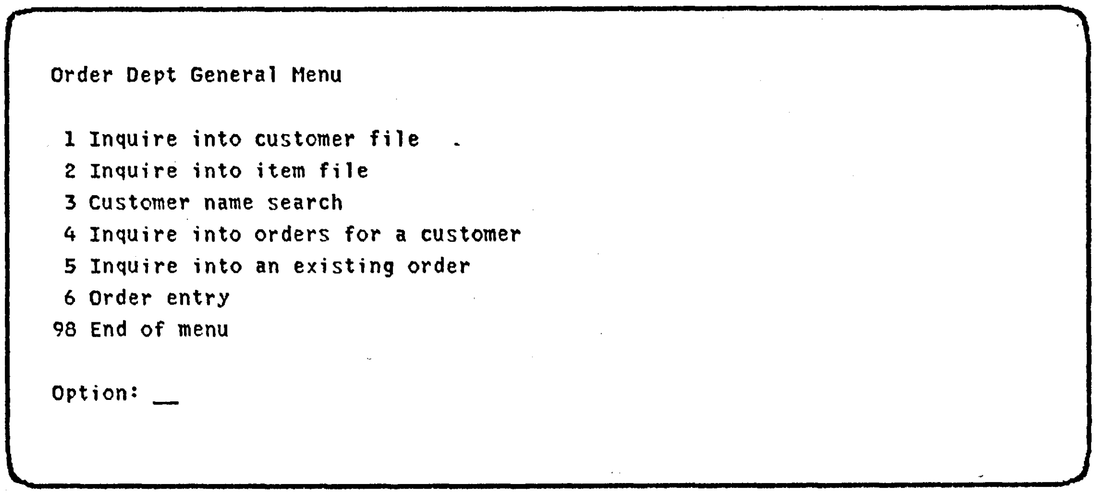
- The following commands can be used only in control language programs:
  - Cancel Receive (CNLRCV)
  - Change Variable (CHGVAR)
  - Declare (DCL)
  - Declare Data Area (DCLDTAARA)
  - Declare File (DCLF)
  - Do (DO)
  - Else (ELSE)
  - End Do (ENDDO)
  - End Program (ENDPGM)
  - Go To (GOTO)
  - If (IF)
  - Monitor Message (MONMSG)
  - Program (PGM)
  - Receive Data Area (RCVDTAARA)
  - Receive File (RCVF)
  - Receive Message (RCVMSG)
  - Remove Message (RMVMSG)
  - Retrieve Job Attributes (RTVJOBATR)
  - Retrieve Message (RTVMSG)
  - Retrieve System Value (RTVSYSVAL)
  - Send Data Area (SNDDTAARA)
  - Send File (SNDF)
  - Send Program Message (SNDPGMMSG)
  - Send Reply (SNDRPY)
  - Send/Receive File (SNDRCVF)
  - Transfer Control (TFRCTL)
  - Wait (WAIT)

- The following commands cannot be used in control language programs:
  - Data (DATA)
  - Display Service Status (DSPSRVSTS)
  - Display Spooled File (DSPSPLF)
  - End Job (ENDJOB)
  - Job (JOB)
  - Prepare APAR (PRPAPAR)

Writing a Program to Control a Menu

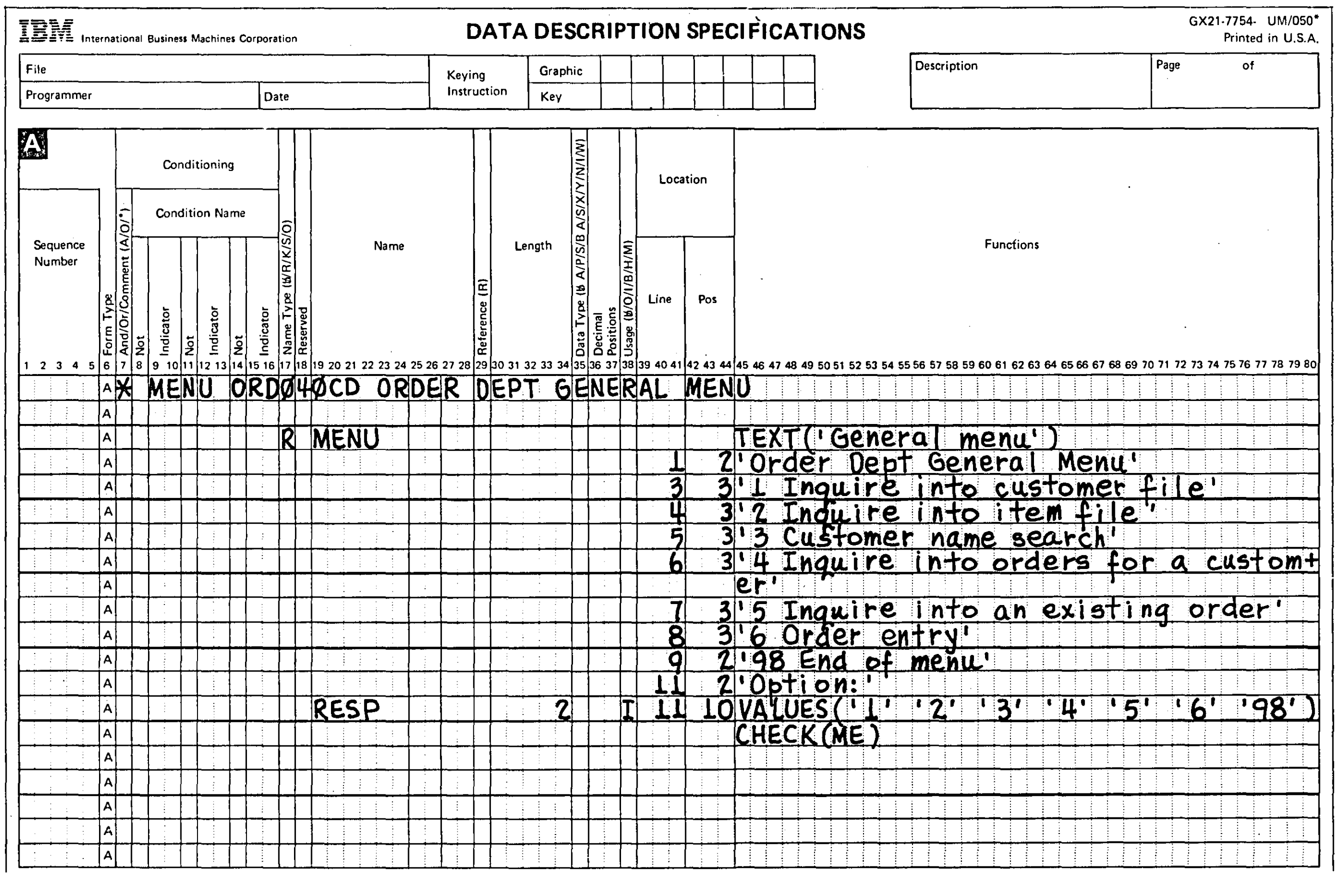
In this example you write a control language program, ORD040C, that controls the displaying of the order department general menu and determines which HLL program to call based on the option selected from the menu.

The order department general menu looks like this:



```
Order Dept General Menu

 1 Inquire into customer file
 2 Inquire into item file
 3 Customer name search
 4 Inquire into orders for a customer
 5 Inquire into an existing order
 6 Order entry
98 End of menu

Option: __
```

The data description specifications (DDS) for the display file look like this:



IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

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

```
     A* MENU ORD040CD ORDER DEPT GENERAL MENU
     A
     A          R MENU                      TEXT('General menu')
     A                                  1  2'Order Dept General Menu'
     A                                  3  3'1 Inquire into customer file'
     A                                  4  3'2 Inquire into item file'
     A                                  5  3'3 Customer name search'
     A                                  6  3'4 Inquire into orders for a custom+
     A                                      er'
     A                                  7  3'5 Inquire into an existing order'
     A                                  8  3'6 Order entry'
     A                                  9  2'98 End of menu'
     A                                 11  2'Option:'
     A            RESP           2   I 11 10VALUES('1' '2' '3' '4' '5' '6' '98')
     A                                      CHECK(ME)
     A
     A
     A
     A
     A
```

The source program for ORD040C looks like this:

```
        PGM /*ORD040C Order Dept General Menu*/
        DCLF FILE(ORD040CD)
START:  SNDRCVF RCDFMT(MENU)
        IF (&RESP=1) THEN(CALL CUS210)
        /*Customer inquiry*/
        ELSE
          IF (&RESP=2) THEN(CALL ITM210)
          /*Item inquiry*/
          ELSE
            IF (&RESP=3) THEN(CALL CUS220)
            /*Cust name search*/
            ELSE
              IF (&RESP=4) THEN(CALL ORD215)
              /*Orders by cust*/
              ELSE
                IF (&RESP=5) THEN(CALL ORD220)
                /*Existing order*/
                ELSE
                  IF (&RESP=6) THEN(CALL ORD410C)
                  /*Order entry*/
                  ELSE
                    IF (&RESP=98) THEN(RETURN)
                    GOTO START
        ENDPGM
```

The DCLF (Declare File) command indicates which file contains the field attributes the system needs to format the order department general menu when the SNDRCVF command is executed. The system automatically declares a variable for each field in the record format in the specified file. The variable name for each field automatically declared is an ampersand (&) followed by the field name. For example, the variable name of the response field RESP in ORD040C is &RESP. There can only be one DCLF command in each program.

The SNDRCVF (Send/Receive File) command is used to send the menu to the display and to receive the option selected from the display.

If the option selected from the menu is 98, ORD040C returns to the program that called it.

Writing a Program to Simplify Work Station Operations

In this example you write a control language program, PARTPGM, to update a parts file. A work station user updates a part in a parts file by entering the part number and the number of parts to add or subtract.

The source program for PARTPGM looks like this:

```
       PGM &PARTFILE
       /*PARTPGM Parts file update program*/
       DCLF FILE(PARTS) RCDFMT(RCD1 RCD2)
       DCL VAR(&PARTFILE) TYPE(*CHAR) LEN(37)
       /*Parameter variable*/
       DCL VAR(&QOH) TYPE(*CHAR) LEN(5)
       /*Quantity on hand*/
       DCL VAR(&X) TYPE(*LGL)
LOOP:  SNDRCVF DEV(PARTS) RCDFMT(RCD1)
       IF (&PARTNBR≤0) THEN(GOTO OUT)
       CALL PARTUPDT (&PARTFILE &PARTNBR &QTY &QOH  &X)
       IF &X THEN(CHGVAR &IN32 (1))
       SNDF DEV(DSP1) RCDFMT(RCD2)
       CHGVAR &IN32 (0)
       GOTO LOOP
OUT:   RETURN
       ENDPGM
```

The DCLF command indicates that the record formats RCD1 and RCD2 in the file PARTS contain the field attributes the system needs to format the part number prompt when the SNDRCVF command is executed. (The record formats are defined through DDS.) The variables &PARTNBR and &QTY are variables automatically declared for the fields PARTNBR and QTY in the record format RCD1. The variable &IN32 is a variable automatically declared for the indicator 32 in the file. Indicator 32 conditions the displaying of RCD2. The three DCL (Declare) commands declare three variables–&PARTFILE, &QOH, and &X–to the system.

If the part number is less than or equal to zero, the program returns control to the program that called it. If the part number is greater than zero, the HLL program PARTUPDT is called and the variables &PARTFILE, &PARTNBR, &QTY, &QOH, and &X are passed to it. PARTUPDT performs the necessary operations to update the number of parts on hand. Within PARTUPDT the variables &QTY and &QOH are compared to determine if there is enough quantity on hand to fill the order. If not, the value of &X is set to 1. When control returns to PARTPGM, &X is tested (IF command) to determine if quantity ordered exceeded quantity on hand. If so, indicator 32 is set to 1 and record format RCD2 is displayed. RCD2 contains a message indicating the condition. After the message is displayed and the user performs any necessary action, indicator 32 is reset to 0 and the part number prompt is displayed again.

Sending and Receiving Information from Data Areas

A data area is an object used to communicate data such as control language variable values between jobs and between programs within a job. (Normally, parameters are used to pass information between programs within the same job.) For example, you can pass a customer number or part number to a program in a different job through a data area. See Chapter 11, *Data Areas* for more information about creating, changing, and deleting data areas. The following paragraphs describe sending and receiving data areas in control language programs.

When you use a data area in a program, you must declare it to the program. When the data area is declared, a control language variable is automatically declared with the same attributes that the data area has. (The name of the CL variable is the data area name preceded by &.) Data can then be moved between the data area and the CL variable. The Send Data Area (SNDDTAARA) command moves the contents of the CL variable to the data area. The RCVDTAARA command moves the contents of the data area to the CL variable for use by the program.

In the following example the contents of the data area INVCMPLT are moved into the CL variable &INVCMPLT. INVCMPLT is tested through the use of an IF command to determine if invoicing is complete.

```
DCLDTAARA INVCMPLT
RCVDTAARA INVCMPLT
/*Check for completion of invoicing*/
IF &INVCMPLT=1
   DO
      CALL CUS680
      CALL CUS681
      CALL CUS0682
   ENDDO
ELSE
  .
  .
  .
```

Retrieving System Values

A system value contains control information for the operation of certain parts of the system. See Chapter 17, *System Values* for a list of the IBM-supplied system values and how you can change and display them. The following paragraphs describe retrieving system values for use in control language programs.

To retrieve a system value for use in a control language program, you place the value in a control language variable using the Retrieve System Value (RTVSYSVAL) command. The value and the variable must be of the same type (character, decimal, or logical) and length.

In the following example the system value QDATSEP (date separator) is moved into the CL variable &DATSEP, which was declared in the program.

```
DCL &DATSEP *CHAR 6
RTVSYSVAL SYSVAL(QDATSEP) RTNVAR(&DATSEP)
```

Monitoring for Messages in Programs

Using the Monitor Message (MONMSG) command, you can monitor for certain types of messages sent to programs. Messages can be monitored according to the command that results in the message being sent or according to the program in which the message is sent. If you monitor on a command, the MONMSG command cannot follow a DO, RETURN, ENDDO, IF, ELSE, GOTO, or ENDPGM command. If you monitor for messages over an entire program, the MONMSG command must immediately follow the PGM command or the last declare command.

For more information about monitoring messages, see Chapter 12, *Message Handling*.

Retrieving Job Attributes

You can retrieve the following job attributes and place their values in a control language variable.

- Job date
- Job switches
- User name for job

You can use these attributes to control your applications. The formats of these job attributes are:

- Job date. A six-character value whose format is defined by the system value QDATFMT.
- Job switches. An eight-character value; each character (switch) is either a 1 or a 0.
- User name. A 10-character value.

To retrieve job attributes, you use the Retrieve Job Attribute (RTVJOBATR) command. You can only retrieve one job attribute in each RTVJOBATR command.

In the following control language program, a RTVJOBATR command retrieves the name of the user who called the program.

```
          PGM
          /*ORD410C Order entry program */
          DCL &CLKNAM TYPE(*CHAR) LEN(10)
          DCL &NXTPGM TYPE(*CHAR) LEN(3)
          •
          •
          •
          RTVJOBATR JOBA(*USER) RTNVAR(&CLKNAM)
BEGIN:CALL ORD410S2 PARM(&NXTPGM &CLKNAM)
          /*Customer prompt*/
          IF(&NXTPGM *EQ 'END') THEN(RETURN)
          •
          •
          •
```

The variable *CLKNAM, in which the user name is to be passed, is first declared using a DCL command. The RTVJOBATR command follows the declare commands. When the program ORD410S2 is called, two variables, &NXTPGM and &CLKNAM, are passed to it.

COMPILING CONTROL LANGUAGE PROGRAMS

A control language source program must be compiled before it can be executed. When you compile a control language source program you can specify

- Whether a compiler listing is to be generated
- Whether space is to be reserved in the compiled program for a patch area
- Whether the program should operate under the program owner's user profile

A control language program can execute using either the owner's user profile or the user's user profile. See *Running a Program Under an Owner's User Profile* in Chapter 18, *Security* for more information.

To compile a control language program, you use the Create Control Language Program (CRTCLPGM) command. The following CRTCLPGM command compiles the program ORD040C and places it in DSTPRODLB.

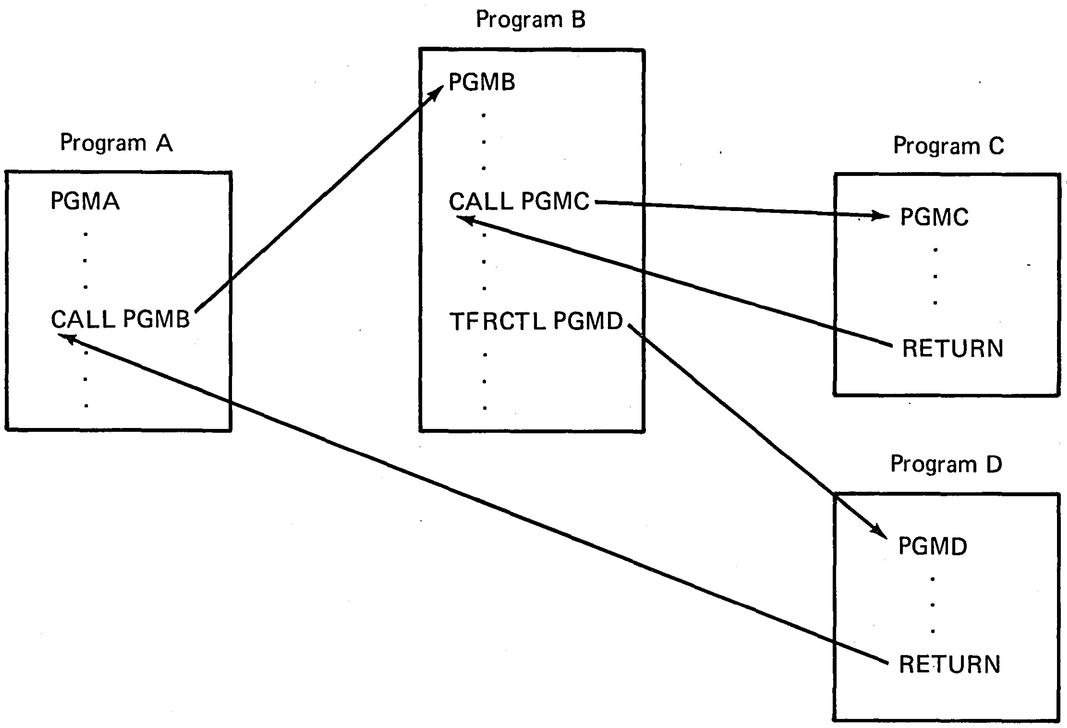
```
CRTCLPGM  PGM(ORD040C.DSTPRODLB)  SRCFILE(QCLSRC)
     TEXT('Order dept general menu program')
```

The source program for ORD040C is in the source file QCLSRC. By default, a compiler listing is generated.

INVOKING CONTROL LANGUAGE PROGRAMS

To invoke a control language program, you can use a CALL command, a Transfer Control (TFRCTL) command, or a command you create. (See Chapter 13, *Defining Commands*.)

When a CALL command is used, the program called returns control to the program containing the CALL command. When a TFRCTL command is used, control returns not to the program containing the TFRCTL command but to the statement after the most recently executed CALL in the program that invoked the transferring program.



When you use either a CALL or TFRCTL command, you can specify parameter values on the command to be passed to the program being called. For the CALL command, these values can be constants or variables. For the TFRCTL command, these values are variables received as parameters.

The parameter values in the CALL and TFRCTL commands must have the same order as the corresponding parameters of the called program. In a CL program, the parameters are specified in the order they appear on the PGM command. In addition, the parameters must be of the same length and type. For example, you write the following control language program:

```
PGM PARM(&P1 &P2)/*PROG*/
DCL VAR(&P1) TYPE(*CHAR) LEN(32)
DCL VAR(&P2) TYPE(*DEC) LEN(15 5)
IF (&P1 *EQ 'DATA') THEN(CALL MYPROG &P2)
ENDPGM
```

The source program is placed in a member in the IBM-supplied source file QCLSRC. The member name is the same as the program name.

You compile this program (the member name is the same as the program name, so SRCMBR need not be specified):

```
CRTCLPGM PGM(PROG) SRCFILE(QCLSRC)
```

Then, when you want to call this program, you specify:

```
CALL PROG (DATA 136)
```

DATA is referenced by the variable &P1; 136 is referenced by the variable &P2.

COMMAND LIST

This is a list of commands related to control language programs. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL.*

Program Commands

Descriptive Name	**Command Name**	**Function**
Create Control Language Program	CRTCLPGM	Creates (compiles) a control language program.
Delete Program	DLTPGM	Deletes a program.
Call	CALL	Invokes a program.

Commands Within Programs

Descriptive Name	**Command Name**	**Function**
Program	PGM	Indicates the start of control language program source.
End Program	ENDPGM	Indicates the end of control language program source.
Declare	DCL	Declares a program variable to a program.
Change Variable	CHGVAR	Changes the value of a control language program variable.
If	IF	Executes commands based on the result of a logical expression.
Else	ELSE	Defines the action to be taken for the else (false) condition of an IF command.

Commands Within Programs (continued)

Descriptive Name	Command Name	Function
Do	DO	Indicates the start of a do group.
End Do	ENDDO	Indicates the end of a do group.
Go To	GOTO	Branches to another command.
Declare File	DCLF	Declares a file to a program.
Send File	SNDF	Writes a record to a display device file.
Send/Receive File	SNDRCVF	Writes a record to a display device file and receives a reply.
Receive Data Area	RCVDTAARA	Moves the contents of a data area to a control language variable.
Receive File	RCVF	Reads a record from a display device file.
Cancel Receive	CNLRCV	Cancels a request for input previously issued by a RCVF or SNDRCVF command.
Wait	WAIT	Waits for a RCVF or SNDRCVF command to be executed.
Declare Data Area	DCLDTAARA	Declares a data area to a program.
Send Data Area	SNDDTAARA	Moves the contents of a control language variable to a data area.
Return	RETURN	Returns to the command following the command that caused a program to be executed (called the program).

Commands Within Programs (continued)

Descriptive Name	Command Name	Function
Transfer Control	TFRCTL	Transfers control to a program.
Monitor Message	MONMSG	Monitors for escape and notify messages sent to a program's program message queue.
Retrieve System Value	RTVSYSVAL	Retrieves a system value and places it into a control language variable.
Retrieve Job Attributes	RTVJOBATR	Retrieves the value of one of the job attributes date, switches, or user name and places the value in a CL variable.

Other Commands

This is a list of commands that are also related to control language programs in general but are not part of the functions presented in this chapter.

Descriptive Name	Command Name	Chapter
Send Program Message	SNDPGMMSG	Chapter 12, Message Handling
Receive Message	RCVMSG	Chapter 12, Message Handling
Send Reply	SNDRPY	Chapter 12, Message Handling
Retrieve Message	RTVMSG	Chapter 12, Message Handling
Display Program References	DSPPGMREF	Chapter 14, Application Documentation
Display Data Base Relations	DSPDBR	Chapter 14, Application Documentaton
Display File Description	DSPFD	Chapter 14, Application Documentation
Display File Field Description	DSPFFD	Chapter 14, Application Documentation

Chapter 5. Data Base

You can create a data base in which the same data is used by many programs in different ways. This can be done by creating many logical files over a single physical file. Logical files can access the data in the physical file in different orders with different attributes or selectively exclude data.

A file must be defined to the system (that is, created) before you can use it in your programs. Records in data base files can be described in two ways:

- Externally described. The fields are described using DDS (data description specifications) and the field attributes are known to the data base.

- Program described. The fields are described in the program that processes the file. DDS are not used to define the fields. The record appears to the data base as one field.

Note: If you use a program described data base file, the field descriptions in your program must be compatible with the field attributes in your data base file.

To use an externally described file in a program, you must specify in the program the name of the file and the member, if more than one, to be processed. (A member is a group of records. For more information see *Adding Members to Files* in this chapter.) When the application program is compiled, the compiler extracts the file description and it becomes part of the compiled program. The application program must open the file before issuing get (read), and put (write) requests to the file. An open request connects a file to a program and a close request disconnects a file from a program.

A data base file contains a file description. The file description contains:

- A description of the record format, which is a description of the fields and the order of the fields in the file's records

- A description of the access path, which is a description of the order in which records are to be retrieved

- Where the data associated with the file is, which is the location of the data for a logical file and allocation parameters for a physical file

This information is specified through DDS (data description specifications) and a create file command. The DDS describes the record format, access paths, and part of where the data is located. The create file command contains the remainder of where the data is located (DTAMBRS parameter for logical file members and allocation parameters for physical files), and additional file attributes.

RECORD FORMAT

The record format describes the fields in the records of a file and the order of the fields in the records. The field description includes the field name, data type (binary, packed decimal, zoned decimal, or character), and length (including decimal positions). The following example shows the relationship between the record format specifications and the records in a file.

Record Format Specifications:

Field	**Description**
ITEMNO	Numeric, six digits, no decimal positions
ITEMDC	Character, 20 positions
PRICE	Numeric, six digits, two decimal positions

Records:

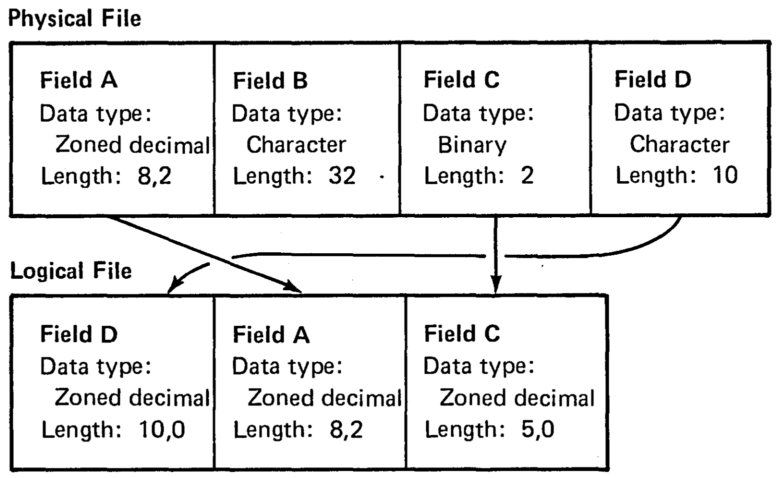
```
ITEMNO        ITEMDC          PRICE
354068HAMMERƀƀƀƀƀƀƀƀƀƀƀƀƀƀ001486
922011SCREWDRIVERƀƀƀƀƀƀƀƀƀ000649
```

For logical files, the record format describes the needed transformation of data from its physical representation in the data base to the format defined in the logical file. For a physical file, the data is actually stored according to the specifications of the record format. A physical file can have only one record format. That is, all records in a physical file are fixed length and have the same field descriptions.

A logical file contains no data. Logical files are used to order data from one or more physical files into different organizations for different uses. If more than one physical file is referenced by a logical file, more than one record format can be specified so that different data records (from different physical files) with varying lengths can be processed.

The following example shows the relationship of the record formats for a physical file and a logical file. In this example, a program needs to have the fields in the physical file changed for the logical file it uses. The changes made are:

- The fields are placed in a different order.
- The fields in the logical file format are a subset of the fields in the physical file format.
- The data types are changed for some fields.
- The field lengths are changed for some fields.



The data from the physical file is transformed to the format used by the logical file so that the data can be sent to the program in a different order for the program's use.

ACCESS PATH

An access path describes the order in which records are to be retrieved. Records in a physical or logical file can be retrieved using an arrival sequence access path or a keyed sequence access path. For logical files, you can also select and omit records based on the value of one or more fields in each record. (This is specified through DDS.)

Arrival Sequence Access Path

The arrival sequence access path is based on the order in which the records are stored in the file. For retrieval or updating, records can be accessed:

- Sequentially, where each record is taken from the next sequential position in the file.

- Directly by relative record number, where the record is identified by its position from the beginning of the file.

An arrival sequence access path is valid only for the following:

- One physical file member

- A logical file in which each member of the logical file is based on only one physical file member. An arrival sequence access path does not span the members in a file. The path applies to each member individually.

(For a discussion of members, see *Adding Members to Files*.)

Keyed Sequence Access Path

For a keyed sequence access path, the sequence of the records in a file are based on the contents of the key fields as defined in DDS. This type of access path is updated whenever records are added, deleted, or modified and the contents of a key field changes. The keyed sequence access path is valid for any data base file. The sequencing of the records in the file is defined when the file is created and is automatically maintained for the file by data base data management.

A key field is a field whose contents are used to access records in a file in a defined sequence. The records in the file can be in either ascending or descending sequence. Consider the following records:

Fields

Record	**EMPNBR**	**CLSNBR**	**CLSNME**	**CPDATE**
1	56218	412	WELDING I	032178
2	41322	412	WELDING I	011378
3	64002	412	WELDING I	011378
4	23318	412	WELDING I	032178
5	41321	412	WELDING I	051878
6	62213	412	WELDING I	032178

If EMPNBR is the key field, there are two possibilities for sequencing these records:

1. If the access path uses EMPNBR as the key field in ascending sequence, the order of the records in the access path is:

   Record 4, Record 5, Record 2, Record 1, Record 6, Record 3

2. If the access path uses EMPNBR as the key field in descending sequence, the order of the records in the access path is:

   Record 3, Record 6, Record 1, Record 2, Record 5, Record 4

You can use more than one key field to sequence a file. Both key fields do not have to use the same order of sequencing. That is, when you use two key fields, one field can use ascending sequence while the other uses descending sequence. Consider the following records:

Fields

Record	**ORDER**	**ORDATE**	**LINE**	**ITEM**	**QTYORD**	**EXTENS**
1	52218	063078	01	88682	425	031875
2	41834	062878	03	42111	30	020550
3	41834	062878	02	61132	4	021700
4	52218	063078	02	40001	62	021700
5	41834	062878	01	00623	50	025000

If the access path uses ORDER, then LINE as the key field, both in ascending sequence, the order of the records in the access path is:

Record 5, Record 3, Record 2, Record 1, Record 4

If the access path uses the key field ORDER in ascending sequence, then ITEM in descending sequence, the order of the records in the access path is:

Record 2, Record 3, Record 5, Record 4, Record 1

A file can be defined as having unique key values. To do so, use the UNIQUE DDS keyword. To define the retrieval order of records in a physical file member that has duplicate key values, use the LIFO (last-in-first-out) DDS keyword or use the default of FIFO (first-in-first-out). The records with duplicate keys can be retrieved in arrival order or in reverse order.

The number of fields that make up a key is restricted in that the total key length cannot exceed 120 bytes.

In a logical file with more than one record format you can use multiple key fields to merge the records of the different formats. Each record format does not have to contain every key field in the key. Consider the following records:

Header Record Format:

	Fields		
Record	**ORDER**	**CUST**	**ORDATE**
1	41882	41394	052478
2	32133	28674	060278

Detail Record Format:

			Fields		
Record	**ORDER**	**LINE**	**ITEM**	**QTYORD**	**EXTENS**
A	32133	01	46412	25	125000
B	32133	03	12481	4	001000
C	41882	02	46412	10	050000
D	32133	02	14201	110	454500
E	41882	01	08265	40	008000

If the access path uses ORDER as the first key field and LINE as the second key field, both in ascending sequence, the order of the records in the access path is:

Record 2, Record A, Record D, Record B, Record 1, Record E, Record C

DATA LOCATION

For a physical file, the description of the location of the data describes how the data is stored in the physical file, which includes how large each member is and, optionally, where the data is stored. For a logical file, the description identifies the physical files from which data is used for the logical file. This information is defined in a create file command, either Create Physical File (CRTPF) or Create Logical File (CRTLF). (For a logical file, this information is also taken from the PFILE keyword in the DDS for the file.) See *Adding Members to Files*, in this chapter, for specific data location information.

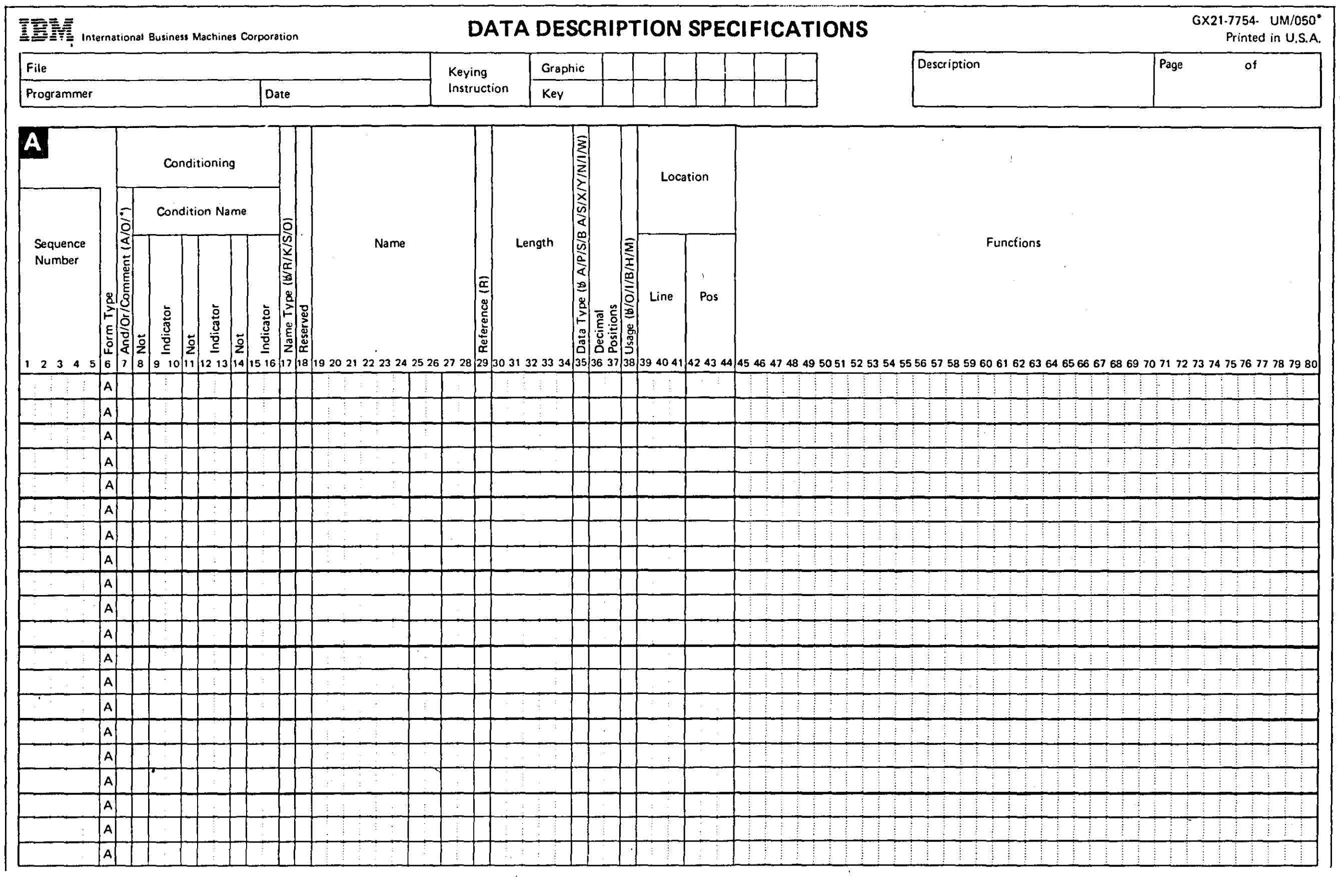
NAME RESTRICTIONS

The file name, record format name, and field names can be as long as 10 characters and must follow all CPF naming conventions (see *How to Specify Names* in Chapter 2, *Control Language*), but you should limit them to the number of characters allowed by the high-level language (HLL) you are using; otherwise, other names must be equated to them in the HLL program. Also, these names should follow any other requirements of the high-level language.

Field names must be unique within a record format. Field names created as a result of concatenating fields or renaming fields cannot be used as select/omit fields or key fields. Record format names and member names must be unique within a file; file names must be unique within a library.

DATA DESCRIPTION SPECIFICATIONS

Externally described data files are described using DDS (data description specifications). DDS is used to name record formats, name and describe fields, and designate key fields. The following shows the DDS form. Note that specific positions are designated for name type (position 17), record format and field names (positions 19 through 28), field length (positions 30 through 34), data type (position 35), and decimal positions (positions 36 and 37).



IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

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

File		Keying Instruction	Graphic	Description	Page of
Programmer	Date		Key		

A

Sequence Number	Form Type	Conditioning							Name Type (ƀ/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (ƀ A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (ƀ/O/I/B/H/M)	Location		Functions
		And/Or/Comment (A/O/*)	Condition Name														Line	Pos	
			Not	Indicator	Not	Indicator	Not	Indicator											
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

Comments can be used in DDS. An * (asterisk) in position 7 of the form indicates that a line is a comment. Comments can be written only in positions 8 through 80.

Keywords on the DDS form let you supply information that is specified less frequently than the information supplied in specific positions on the form. For example, the keyword DESCEND changes the sequencing of records from ascending (the normal sequence used by the system) to descending for a particular key field. Keywords are specified in the functions positions (positions 45 through 80).

A complete list of keywords is in the section *Keyword Summary* in this chapter.

Before DDS can be used to create a file, it must be placed in a source file. See Chapter 10, *Source Files* for how source is placed in source files.

For complete information about DDS, see the *CPF Reference Manual—DDS*.

CREATING A PHYSICAL FILE

A physical file contains data stored in fixed-length records and has only one record format. Each record in a physical file matches the field descriptions in the DDS for the physical file.

Records are stored in a physical file in the order in which they are placed in the file, that is, in arrival sequence. However, the records can be retrieved in any order specified by a valid access path.

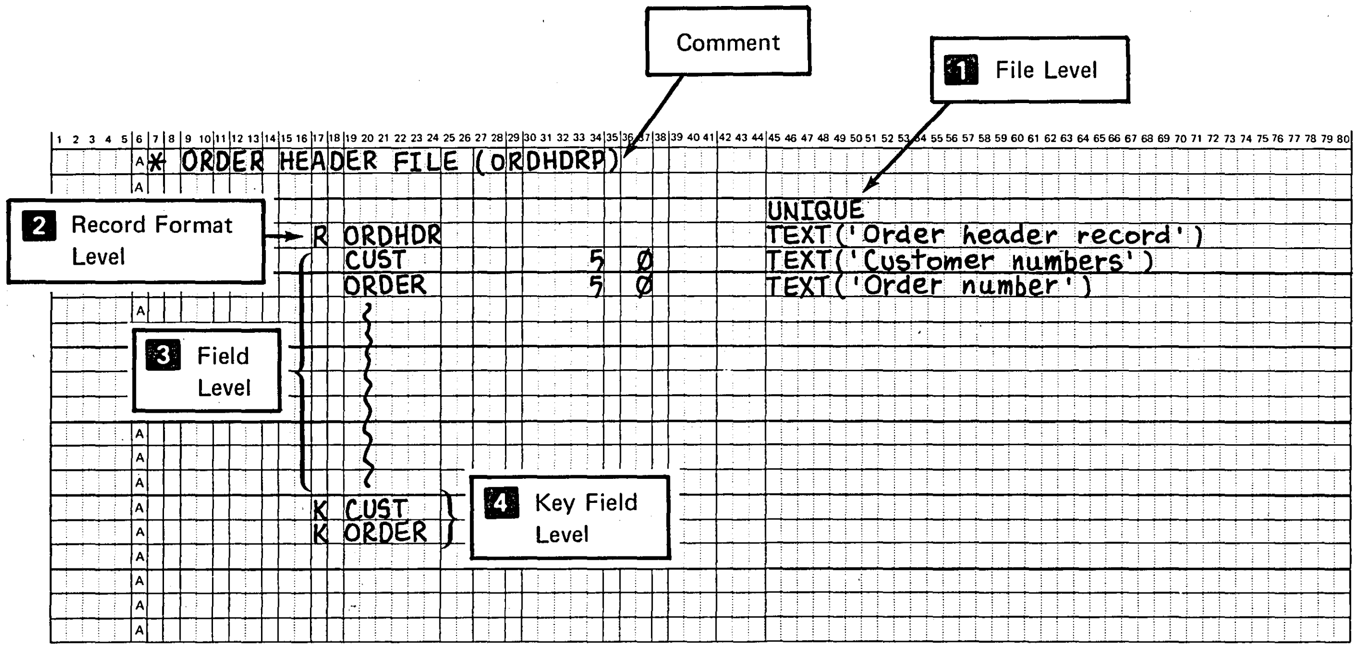
The DDS for a physical file must be in the following order (Figure 2):

1 File level keywords

2 Record format name and, for a newly described record format, text description

3 For a newly described record format, name and description of each field

4 Key fields



```
     A* ORDER HEADER FILE (ORDHDRP)
     A
     A                                      UNIQUE
     A          R ORDHDR                    TEXT('Order header record')
     A            CUST           5  0       TEXT('Customer numbers')
     A            ORDER          5  0       TEXT('Order number')
     A
     A
     A
     A
     A          K CUST
     A          K ORDER
     A
     A
     A
     A
```

Figure 2. Order of DDS for a Physical File

In the following example, you are creating a physical file ORDHDRP (an order header file) that has an arrival sequence access path. Figure 3 describes the fields in the records and shows how they are coded in DDS.

Record Format (ORDHDR):

Customer Number (CUST)	Order Number (ORDER)	Order Date (ORDATE)	Purchase Order Number (CUSORD)	Ship Code (SHPVIA)	Order Status (ORDSTS)	•••
5 bytes Numeric (packed) No decimals	5 bytes Numeric (packed) No decimals	6 bytes Numeric (packed) No decimals	15 bytes Numeric (packed) No decimals	15 bytes Character	1 byte Character	

```
     A* ORDER HEADER FILE (ORDHDRP)
     A          R ORDHDR                    TEXT('Order header record')
     A            CUST           5  Ø
     A            ORDER          5  Ø
     A            ORDATE         6  Ø
     A            CUSORD        15  Ø
     A            SHPVIA        15
     A            ORDSTS         1
     A            OPRNME        1Ø
     A            ORDAMT         9  2
     A            CUTYPE         1
     A            INVNBR         5  Ø
     A            PRTDAT         6  Ø
     A            SEQNBR         5  Ø
     A            OPNSTS         1
     A            LINES          3  Ø
     A            ACTMTH         2  Ø
     A            ACTYR          2  Ø
     A            STATE          2
     A
```

Figure 3. DDS for a Physical File

The R in position 17 indicates that you are defining a record format. The record format name ORDHDR is specified in positions 19 through 28. You make no entry in this position when you are describing a field; a blank indicates field.

If the data type (position 35) is not specified, the decimal positions entry is used to determine the data type. If the decimal positions (positions 36 through 37) are blank, the data type is assumed to be character (A); if these positions contain a number 0 through 31, the data type is assumed to be packed decimal (P).

If a packed or zoned decimal field is to be used in an HLL program, the field length must be limited to the length allowed by the high-level language you are using. The length is not the length of the field in storage but the number of digits or characters specified externally from storage. For example, if a field's length in storage is 3 in packed form and its length is 5 in unpacked form, 5 is specified as the field's length in DDS.

Note: System/38 performs arithmetic operations more efficiently for packed decimal than for zoned decimal. Packed is the default for numeric data in data base files.

To actually create the physical file ORDHDRP, you must enter a Create Physical File (CRTPF) command like the following. Your DDS was entered into the IBM-supplied source file QDDSSRC. By default, the system uses QDDSSRC to find your source because the SRCFILE parameter was not specified.

```
CRTPF  FILE(ORDHDRP.DSTPRODLB)
       TEXT('Order header physical file')
```

The file ORDHDRP is placed in the library DSTPRODLB. This file has one member, which is also named ORDHDRP. (See *Adding Members to Files* in this chapter for more information about members.)

CREATING A LOGICAL FILE

A logical file cannot be created unless all physical files on which it is based exist. Record formats in a logical file can be

- A new record format based on a physical file record format
- The same record format as in a previously described physical or logical file

If you use the same format as the associated physical file, a change to the physical file record format requires a change in the logical file record format, and any program using the logical file must be recompiled. If your logical file record format is independent of the physical file record format, a change to the physical file record format might not create a change to the logical file record format. The program using the logical file must be recompiled only if the logical file record format changes.

The three ways of specifying a logical file record format are:

1. Use the PFILE keyword to specify the physical files associated with the logical file and specify only the key fields. The record format of the first physical file specified in PFILE is used. The format name specified must be the same as the format name used in the physical file.

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

2. Use the PFILE keyword to specify the physical file associated with the logical file and specify field names and the key fields if key fields are to be used. The record format is new and not the same as the physical file record format.

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

3. Use the FORMAT keyword to name a previously described record format. The previously described record format can be in a physical or logical file. The format name specified must be the same as the format name used in the file specified by the FORMAT keyword.

```
     A
     A          R CUSRCD                    PFILE(CUSMSTP)
     A                                      FORMAT(CUSMSTL)
     A          K ZIP
     A          K SEARCH
     A
```

The DDS for a logical file must be in the following order (Figure 4):

1 File level keywords

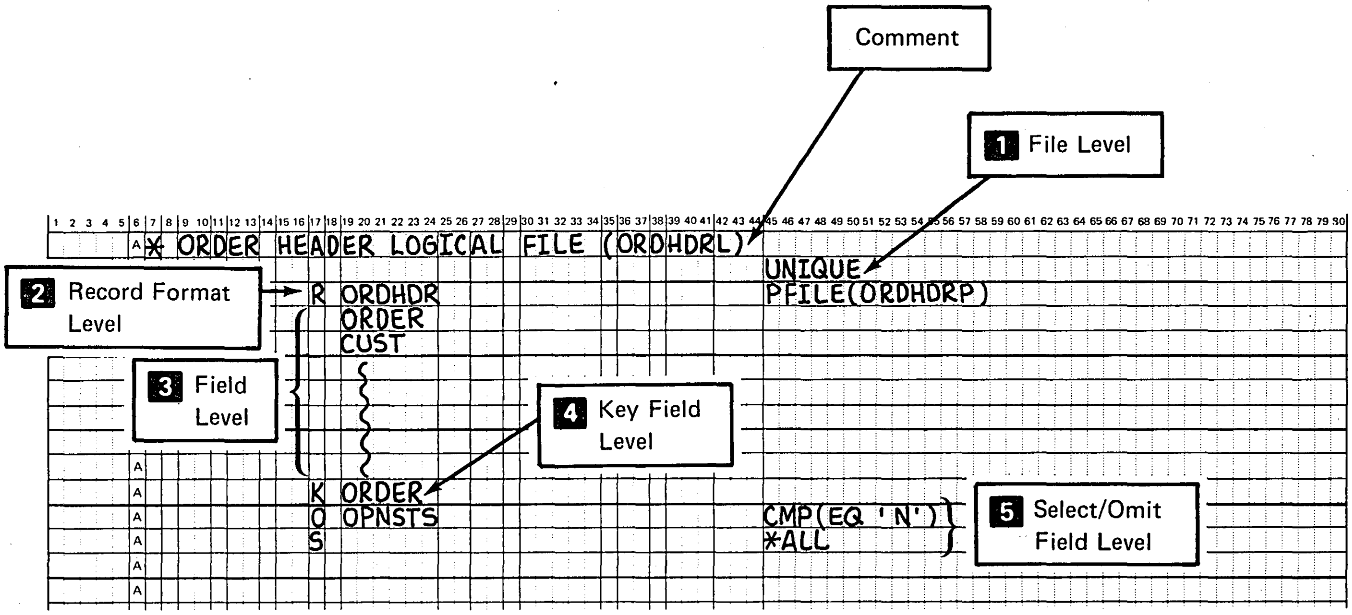
For each record format:

2 Record format name, associated physical file name, and for a newly described record format, optional text description

3 For a newly described record format, name and description of each field

4 Key fields (optional)

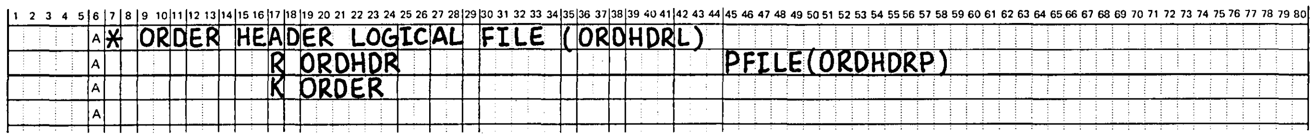
5 Select/omit fields (optional)



```
     A* ORDER HEADER LOGICAL FILE (ORDHDRL)
                                            UNIQUE
                R ORDHDR                    PFILE(ORDHDRP)
                  ORDER
                  CUST
     A
     A          K ORDER
     A          O OPNSTS                    CMP(EQ 'N')
     A          S                           *ALL
     A
     A
```

Figure 4. Order of DDS for a Logical File

In the following example, you create a logical file ORDHDRL (an order header file) that uses the key field ORDER (order number) to define the access path. The record format is the same as the associated physical file ORDHDRP. The record format name for the logical file must be the same as the record format name in the physical file because no field descriptions are given. Figure 5 shows how the DDS for ORDHDRL is coded.



```
     A* ORDER HEADER LOGICAL FILE (ORDHDRL)
     A          R ORDHDR                    PFILE(ORDHDRP)
     A          K ORDER
     A
```

Figure 5. DDS for a Logical File

To actually create the logical file ORDHDRL, you must enter a Create Logical File (CRTLF) command like the following. Your DDS was entered into the IBM-supplied source file QDDSSRC. By default, the system uses QDDSSRC to find your source; the member name is ORDHDRL.

```
CRTLF  FILE(ORDHDRL.DSTPRODLB)
       TEXT('Order header logical file')
```

The file ORDHDRL with one member of the same name is placed in the library DSTPRODLB. (See *Adding Members to Files* in this chapter for more information.)

CREATING A FIELD REFERENCE FILE

To simplify record format descriptions and to ensure that fields are used consistently, you can define all the fields you need for an application or any grouping in one file, a field reference file. The field reference file is a physical file containing no data. You create this file using DDS and the Create Physical File (CRTPF) command.

Once the field reference file has been created, you can build physical formats from this file without describing the characteristics of each field in each file. When you build physical files, all you need do is reference the field reference file (using the REF and REFFLD keywords) and specify any modifications. Any modifications to the field descriptions and keywords specified override the referenced field descriptions. You can also specify an access path for the physical files.

Note: The field reference file can also be referenced to build externally described data device files.

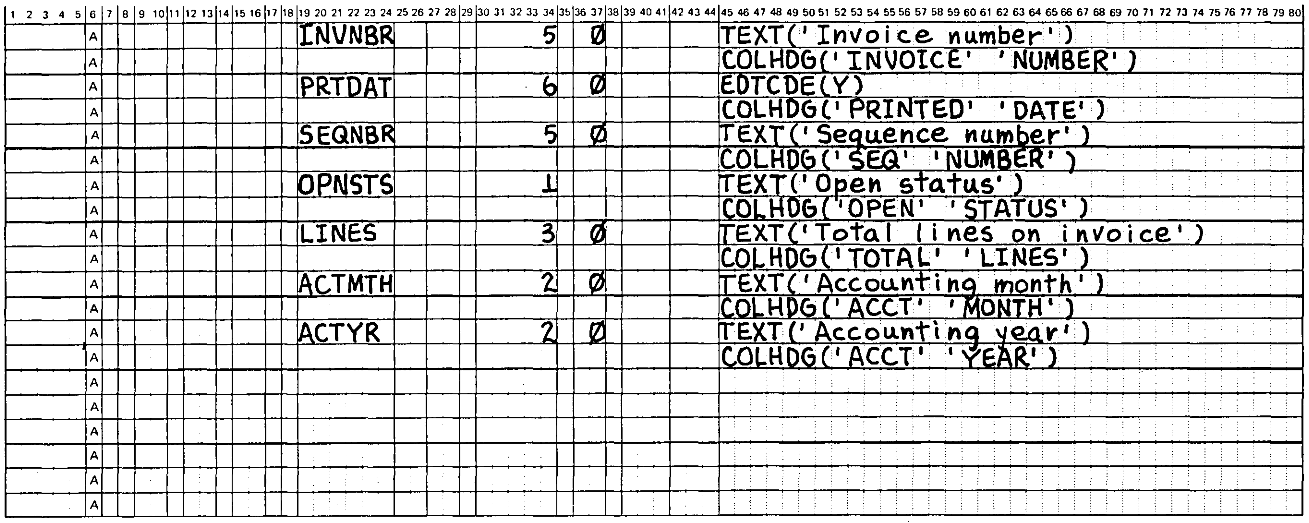
In the following example, you create a field reference file DSTREF for your distribution applications. Figure 6 shows the DDS needed to create DSTREF.

```
     A* FIELD REFERENCE FILE (DSTREF)
     A          R DSTREF                    TEXT('Field reference file')
     A
     A* FIELDS DEFINED BY CUSTOMER MASTER RECORD (CUSMST)
     A            CUST           5  0       TEXT('Customer numbers')
     A                                      COLHDG('CUSTOMER' 'NUMBER')
     A            NAME          20          TEXT('Customer name')
     A            ADDR      R               REFFLD(NAME *SRC)
     A                                      TEXT('Customer address')
     A            CITY      R               REFFLD(NAME *SRC)
     A                                      TEXT('Customer city')
     A            STATE          2          TEXT('State abbreviation')
     A                                      CHECK(MF)
     A            CRECHK         1          TEXT('Credit check')
     A                                      LIST('Y' 'N')
     A            SEARCH         6          TEXT('Customer name search')
     A                                      COLHDG('SEARCH CODE')
     A            ZIP            5  0       TEXT('Zip code')
     A                                      CHECK(MF)
     A            CUTYPE         1          COLHDG('CUSTOMER' 'TYPE')
```

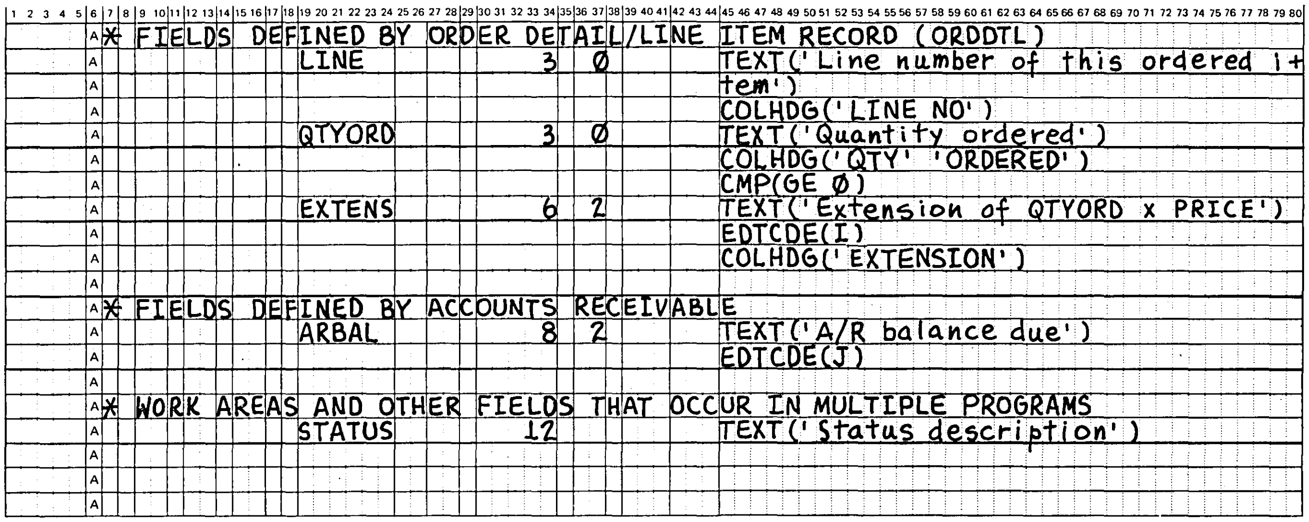
```
     A                                      RANGE(1 5)
     A
     A* FIELDS DEFINED BY ITEM MASTER RECORD (ITMAST)
     A            ITEM           5          TEXT('Item number')
     A                                      COLHDG('ITEM' 'NUMBER')
     A                                      CHECK(M10)
     A            DESCRP        18          TEXT('Item description')
     A            PRICE          5  2       TEXT('Price per unit')
     A                                      EDTCDE(J)
     A                                      CMP(GT 0)
     A                                      COLHDG('PRICE')
     A            ONHAND         5  0       TEXT('On hand quantity')
     A                                      EDTCDE(Z)
     A                                      CMP(GE 0)
     A                                      COLHDG('ON HAND')
     A            WHSLOC         3          TEXT('Warehouse location')
     A                                      CHECK(MF)
     A                                      COLHDG('BIN NO')
     A            ALLOC     R               REFFLD(ONHAND *SRC)
     A                                      TEXT('Allocated quantity')
```

```
     A                                      CMP(GE 0)
     A                                      COLHDG('ALLOCATED')
     A
     A* FIELDS DEFINED BY ORDER HEADER RECORD (ORDHDR)
     A            ORDER          5  0       TEXT('Order number')
     A                                      COLHDG('ORDER' 'NUMBER')
     A            ORDATE         6  0       TEXT('Order date')
     A                                      EDTCDE(Y)
     A                                      COLHDG('DATE' 'ORDERED')
     A            CUSORD        15          TEXT('Customer purchase order numbe+
     A                                      r')
     A                                      COLHDG('P.O.' 'NUMBER')
     A            SHPVIA        15          TEXT('Shipping instructions')
     A            ORDSTS         1          TEXT('Order status code')
     A                                      COLHDG('ORDER' 'STATUS')
     A            OPRNME        10          TEXT('Operator name')
     A                                      COLHDG('OPERATOR NAME')
     A            ORDAMT         9  2       TEXT('Total order value')
     A                                      COLHDG('ORDER' 'AMOUNT')
     A
```

Figure 6 (Part 1 of 2). DDS for a Field Reference File



```
     A            INVNBR         5  Ø       TEXT('Invoice number')
     A                                      COLHDG('INVOICE' 'NUMBER')
     A            PRTDAT         6  Ø       EDTCDE(Y)
     A                                      COLHDG('PRINTED' 'DATE')
     A            SEQNBR         5  Ø       TEXT('Sequence number')
     A                                      COLHDG('SEQ' 'NUMBER')
     A            OPNSTS         1          TEXT('Open status')
     A                                      COLHDG('OPEN' 'STATUS')
     A            LINES          3  Ø       TEXT('Total lines on invoice')
     A                                      COLHDG('TOTAL' 'LINES')
     A            ACTMTH         2  Ø       TEXT('Accounting month')
     A                                      COLHDG('ACCT' 'MONTH')
     A            ACTYR          2  Ø       TEXT('Accounting year')
     A                                      COLHDG('ACCT' 'YEAR')
     A
     A
     A
     A
     A
     A
```



```
     A* FIELDS DEFINED BY ORDER DETAIL/LINE ITEM RECORD (ORDDTL)
     A            LINE           3  Ø       TEXT('Line number of this ordered i+
     A                                      tem')
     A                                      COLHDG('LINE NO')
     A            QTYORD         3  Ø       TEXT('Quantity ordered')
     A                                      COLHDG('QTY' 'ORDERED')
     A                                      CMP(GE Ø)
     A            EXTENS         6  2       TEXT('Extension of QTYORD x PRICE')
     A                                      EDTCDE(I)
     A                                      COLHDG('EXTENSION')
     A
     A* FIELDS DEFINED BY ACCOUNTS RECEIVABLE
     A            ARBAL          8  2       TEXT('A/R balance due')
     A                                      EDTCDE(J)
     A
     A* WORK AREAS AND OTHER FIELDS THAT OCCUR IN MULTIPLE PROGRAMS
     A            STATUS        12          TEXT('Status description')
     A
     A
     A
```

Figure 6 (Part 2 of 2). DDS for a Field Reference File

To actually create the field reference file, you must use a Create Physical File (CRTPF) command like the following. Assume that your DDS was entered into the source file FRSOURCE; the member name is DSTREF.

```
CRTPF  FILE(DSTREF.DSTPRODLB)
       SRCFILE(FRSOURCE.QGPL) MBR(*NONE)
       TEXT('Distribution field reference file')
```

Note: The files used in the remaining sections of this chapter are based on this field reference file.

If the physical file ORDHDRP were built from DSTREF, the DDS would be as shown in Figure 7.

```
     A* ORDER HEADER FILE (ORDHDRP) - PHYSICAL FILE RECORD DEFINITION
     A                                      REF(DSTREF)
     A          R ORDHDR                    TEXT('Order header record')
     A            CUST      R
     A            ORDER     R
     A            ORDATE    R
     A            CUSORD    R
     A            SHPVIA    R
     A            ORDSTS    R
     A            OPRNME    R
     A            ORDAMT    R
     A            CUTYPE    R
     A            INVNBR    R
     A            PRTDAT    R
     A            SEQNBR    R
     A            OPNSTS    R
     A            LINES     R
     A            ACTMTH    R
     A            ACTYR     R
     A            STATE     R
```

Figure 7. DDS for a Physical File (ORDHDRP) Built from a Field Reference File

The REF keyword (positions 45 through 80) with the field reference file name indicates the file from which field descriptions are to be used. The R in position 29 indicates that the field description is to be taken from the reference file and that the fields have the same name.

CREATING A LOGICAL FILE WITH MORE THAN ONE RECORD FORMAT

A logical file can have more than one record format. A logical file with more than one record format lets you use related records (physical files) by referencing only one logical file.

Each record format is always associated with one or more physical files. In a single logical file you can use the same physical file in more than one record format.

In the following example, you are creating a logical file ORDFILL with two record formats. One record format is defined for order header records from the physical file ORDHDRP; the other is defined for order detail records from the physical file ORDDTLP. Figure 8 shows the DDS for the physical file ORDDTLP. (Figure 7 shows the DDS for the physical file ORDHDRP). Figure 9 shows the DDS for the logical file ORDFILL.

```
     A* ORDER DETAIL FILE (ORDDTLP) - PHYSICAL FILE RECORD DEFINITION
     A                                      REF(DSTREF)
     A          R ORDDTL                    TEXT('Order detail record')
     A            CUST      R
     A            ORDER     R
     A            LINE      R
     A            ITEM      R
     A            QTYORD    R
     A            DESCRP    R
     A            PRICE     R
     A            EXTENS    R
     A            WHSLOC    R
     A            ORDATE    R
     A            CUTYPE    R
     A            STATE     R
     A            ACTMTH    R
     A            ACTYR     R
     A
     A
     A
```

Figure 8. DDS for Physical File ORDDTLP

```
     A* ORDER TRANSACTION LOGICAL FILE (ORDFILL)
     A          R ORDHDR                    PFILE(ORDHDRP)
     A          K ORDER
     A
     A          R ORDDTL                    PFILE(ORDDTLP)
     A          K ORDER
     A          K LINE
     A
```

Figure 9. DDS for the Logical File ORDFILL

The logical file record format ORDHDR uses one key, ORDER, for sequencing; the logical file record format ORDDTL uses two keys, ORDER and LINE, for sequencing.

To create the logical file ORDFILL with two associated physical files, use a CRTLF command like the following:

```
CRTLF  FILE(ORDFILL.DSTPRODLB)
       DTAMBRS((ORDHDRP(ORDHDRP)) (ORDDTLP(ORDDTLP)))
       TEXT('Order transaction logical file')
```

The DDS source is in the member ORDFILL in the file QDDSSRC. The file ORDFILL with a member of the same name is placed in the DSTPRODLB library. The access path for the logical file member ORDFILL sequences records from both the ORDHDRP and ORDDTLP files (and members of the same names). Duplicate keys within each physical file are retrieved in FIFO order. Because record formats for both physical files are keyed on ORDER as the common field, and because of the order in which they were specified in the DTAMBRS parameter, they are merged in ORDER sequence with duplicates between files retrieved first from the header file ORDHDRP and second from the detail file ORDDTLP.

When you create a logical file with more than one record format, you might want to write a format selector. The format selector is a program to determine where a record should be placed in the data base. The format selector is used only when no record format name is given by the application program for records that must be inserted into the data base.

When you write a format selector, you should keep in mind the following information:

- A format selector can be a control language program or any high-level language program.

- The format selector cannot be created with the parameter option (*USER specified for the USRPRF parameter) of using the owner's user profile to execute.

- It is not necessary to specify a format selector for a logical file or logical file member with only one record format.

The name of the format selector is specified in the Create Logical File (CRTLF) command, in the FMTSLR parameter.

SELECTING AND OMITTING RECORDS

For a logical file with a keyed sequence access path, you can select and omit records from the file. The select function selects records from a physical file instead of using all physical file records in the logical file. The omit function specifies which records from a physical file are to be omitted from the logical file. Selecting and omitting records is based on comparisons specified in the DDS for the logical file. For example, in a logical file that contains order detail records, you can specify that the only records you want to use are those where the quantity ordered is greater than the quantity shipped. All other records are omitted from the access path. The records remain in the physical file but are not retrieved for the logical file.

Note: Field names used for selecting and omitting records must be physical file field names.

In Figure 10, you select all (ALL keyword) of the records from a record format except those from Department 12 (DPTNBR compared to 12). For Department 12, only those records containing an item number (ITMNBR) of 112505, 428707, or 480100 are selected.

```
     A          S DPTNBR                    CMP(EQ 12)
     A            ITMNBR                    VALUES(1125Ø5 4287Ø7 48Ø1ØØ)
     A          O DPTNBR                    CMP(EQ 12)
     A          S                           ALL
     A
     A
```

Figure 10. DDS for Selecting and Omitting Records

If ALL is not specified as the last select or omit specification, it is automatically generated. If a select was specified last, an omit with ALL is generated; if an omit was specified last, a select with ALL is generated.

SHARING RECORD FORMATS

A record format can be described once and can be used (shared) by many files. The file originally describing the record format can be deleted without affecting the files sharing the record format. But as soon as the last file using the record format is deleted, the record format is deleted.

In Figure 11, you describe a logical file CUSMSTL1 that shares record formats with another logical file CUSMSTL.

```
     A* CUSTOMER NAME SEARCH LOGICAL FILE (CUSMSTL1)
     A          R CUSREC                    PFILE(CUSMSTP)
     A                                      FORMAT(CUSMSTL)
     A          K ZIP
     A          K SEARCH
     A
     A
```

Figure 11. DDS for a File that Shares Record Formats

CUSMSTL1 uses the record format CUSREC from CUSMSTL. The FORMAT keyword indicates that a previously described record format is to be used.

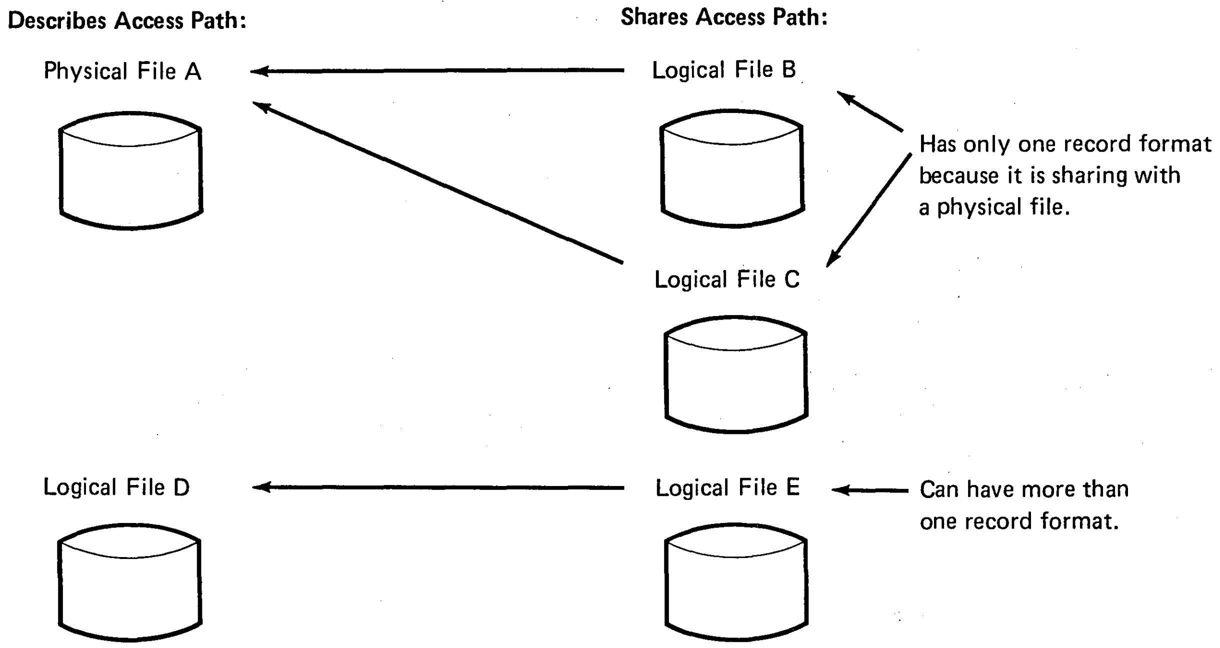
If a logical file is defined but no field descriptions are specified and the FORMAT keyword is not specified, the record format of the first physical file (specified first on the PFILE keyword for the logical file) is automatically shared. You can save time defining a file if you share record formats, but you have less data independence if the shared format is changed. If the based on format is changed (by deleting all related files and creating the based-on file again), it is changed for all files that share it.

SHARING AN ACCESS PATH

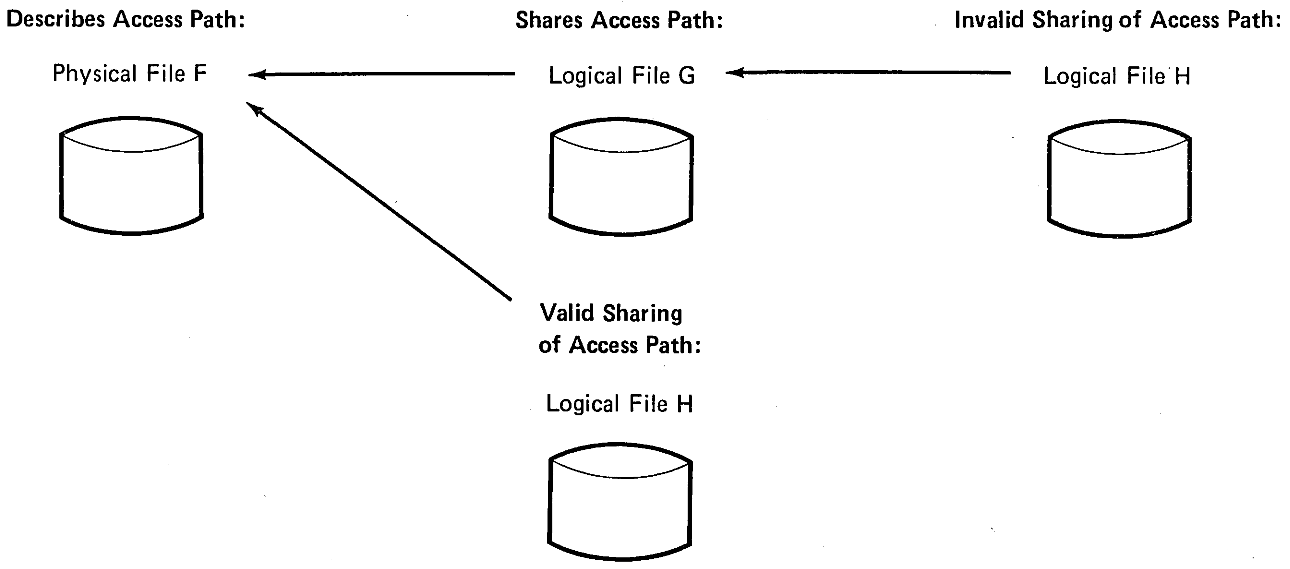
When two or more files use the same data (are based on the same set of physical files) and the same organization of the data, they can share the same keyed sequence access path. When access paths are shared, the amount of system activity required to maintain access paths is reduced. The amount of auxiliary storage used by the files is also reduced. Two users can access the same physical file with different record formats. That is, two users can use the same data and access paths but use different logical files. However, the two formats can be using different fields in the data.

When an access path is shared, the file that references the access path must be a logical file and is dependent on the file (physical or logical) that describes (creates) the access path. The file describing the access path cannot be deleted unless the file referencing the access path is deleted first.

When an access path is shared and the file describing the access path is a physical file, the file referencing (sharing) the access path must be a logical file with only one record format. If the file describing the access path is a logical file, the logical file sharing the access path can contain more than one record format. The following illustrates these concepts.



When an access path is shared, the file describing the access path must not be sharing an access path that is already being shared from another file. That is, there is only one level of access path sharing. The following illustrates how access paths cannot be shared.



When files share an access path, the fields in the file sharing the access path must follow the same rules that are applied to any field description for a logical file.

The file sharing the access path must have the same number of physical files (specified using PFILE keywords) as the file describing the access path, and the physical files must be specified in the same order. The record formats used in the referencing file must contain the key fields and select/omit fields used in the file describing the access path.

The fields in the record formats of the file sharing the access path can be a subset of the fields in the physical file describing the access path, or the record format of the file sharing the access path can include more fields from the physical file record format than are described in the file describing the access path.

The ACCPTHMBR parameter must be specified on the CRTLF command when the file shares an access path (the ACCPTH DDS keyword is used) and a member is to be created:

```
CRTLF  FILE(ORDHDRL1.DSTPRODLB)
       ACCPTHMBR(ORDHDRL)
       TEXT('Order header logical file')
```

The DTAMBRS parameter defaults to the one member of the file ORDHDRP.

In the following example, you are creating a logical file ORDHDRL1 that shares an access path with another logical file ORDHDRL (see *Creating a Logical File*). Figure 12 shows the DDS needed to create ORDHDRL1.

```
     A* ORDER HEADER LOGICAL FILE (ORDHDRL1)
     A                                      ACCPTH(ORDHDRL.DSTPRODLB)
     A          R ORDHDR                    PFILE(ORDHDRP)
     A
     A
```

Figure 12. DDS for a File Sharing Access Paths

CONCATENATING FIELDS

When you concatenate fields, you combine two or more fields from a physical file record format to make one field in a logical file record format. For example, a physical file record format contains the fields ADDR, CITY, and STATE (customer address, city, and state). For a logical file, you concatenate these fields into one field, ADDRESS.

The field length for a concatenated field is the sum of the field lengths of the fields included in the concatenated field. This field length cannot be changed. However, when you concatenate fields, you can specify the following information for the new field.

- Edit code or edit word
- Column headings
- Validity checking data
- Text description

Note: This editing and validity checking information is not used by the data base but is retrieved when field specifications from the data base file are used in a device file.

When fields are concatenated, the data types could change. The following shows what data types result:

Fields Being Concatenated	**Concatenated Field**
Binary	Zoned decimal
Zoned decimal	Zoned decimal
Packed decimal	Zoned decimal
Character	Character
Character and numeric (alphameric)	Character

When numeric fields are concatenated, the sign of the last field is used as the sign of the concatenated field.

Notes:
1. Fields containing nonzero decimal positions cannot be included in a concatenated field.
2. A concatenated field name cannot be used as a key, select, or omit field name.

The following shows the field description in DDS for concatenation. The CONCAT keyword is used to concatenate fields.

```
     A
     A            NAME
     A            ADDRESS                   CONCAT(ADDR CITY STATE)
     A            ZIP
     A
     A
```

HANDLING DUPLICATE KEY VALUES

Duplicate key values are two or more key values that are the same and are entered or passed in the same key field for different records in the same file. Duplicate key values apply across all key fields for a record. For example, if a record format has two key fields ORDER and ORDATE, duplicate key values occur when the combination of these fields is the same in two or more records. Figure 13 shows duplicate key values.

First Key Field	Second Key Field				
ORDER	**ORDATE**	**LINE**	**ITEM**	**QTYORD**	**EXTENS**
41834	062880	03	42111	30	020550
41834	062880	02	61132	4	021700
41834	062880	01	00623	50	025000

Figure 13. Duplicate Key Values

In Figure 13 if either ORDER or ORDATE had varied or if LINE had been used as a third key field, the records would not have had duplicate key values (see Figure 14).

First Key Field	Second Key Field	Third Key Field			
ORDER	**ORDATE**	**LINE**	**ITEM**	**QTYORD**	**EXTENS**
41834	062880	03	42111	30	020550
41834	062880	02	61132	04	021700
41834	062880	01	00623	50	025000

Figure 14. No Duplicate Key Values

You can prevent duplicate key values in your files by specifying that key values must be unique. That is, a record cannot be entered into a file if its key value is the same as the key value of a record already existing in the file. If UNIQUE is specified for a file, no other files sharing data can add records to the file that would result in duplicate keys. Figure 15 shows the DDS for a logical file that requires unique key values.

```
     A* ORDER TRANSACTION LOGICAL FILE (ORDFILL)
     A                                      UNIQUE
     A          R ORDHDR                    PFILE(ORDHDRP)
     A          K ORDER
     A
     A          R ORDDTL                    PFILE(ORDDTLP)
     A          K ORDER
     A          K LINE
     A
     A
```

Figure 15. DDS for Unique Key Values

Note: When UNIQUE is used, you must specify immediate maintenance (on the CRTPF or CRTLF command) for the access path so that attempts to add nonunique keys can be detected and rejected immediately.

If you allow duplicate key values and they occur, you can handle the records with duplicate key values in one of the following orders:

1. First-in-first-out (FIFO). This is the default order for duplicates in one physical file member.

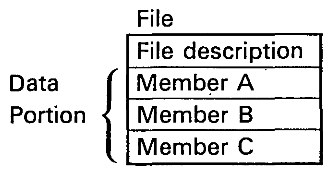
2. Last-in-first-out (LIFO). This can be used for duplicates within one physical file member.

3. If duplicate key values occur within records that are from different physical file members, the system retrieves the records in the order in which the files and members are specified in the DTAMBRS parameter on the CRTLF or ADDLFM command.

If UNIQUE is specified for a logical file, a record with a duplicate key value cannot be added to a physical file that the logical file is based on. For example, two logical files LF1 and LF2 are based on the physical file PF1. UNIQUE is specified for LF1. If you are using LF2 and as a result are adding a record to PF1, you cannot add the record if it causes a duplicate key value in LF1.

ADDING MEMBERS TO FILES

A member is the operational part of a file. If there is more than one member in a file, each member is a subset of the data for the file. The following illustrates the concept of members.



Each member has its associated data and its own access path for that data. The access paths conform to the same rules in the file description.

Before you can perform any input or output (I/O) operations with a file, you must add a member to the file. All I/O requests are made to members. There are two ways to add members to files:

- Add the member when you create the file using the create file command. You can name the member using the MBR parameter or let it default to the same name as the file.
- Add the member after the file is created using the add member (ADDPFM or ADDLFM) command.

Some information about members is specified when the file is created (CRTLF or CRTPF command) instead of when the member is created (ADDPFM or ADDLFM command). This information relates to all members of the file. The following lists the information that is common to all members of both logical and physical files. (Parameters are given in parentheses.)

- *The maximum number of members that can be contained in the file (MAXMBRS parameter).* You can specify a number from one through 32767.
- *The maintenance (immediate or rebuild) of the keyed sequence access paths for the members (MAINT parameter).*

  Once a keyed sequence access path has been created, it must be maintained so that it reflects any change made to the data it is associated with. When a file member is being processed (is open), its access path is maintained as changes are made to the file member. However, because data can be shared by more than one file and more than one access path can exist to the same data, updating data in one file might require changes to be made in other access paths. As changes are made to the data, the access paths for all the related active (open) files on the system are also updated. This is true even though the data is being changed through another file or access path.

The type of access path maintenance to specify depends on the size of the members (number of records) and frequencies of adds, deletes, and updates to a file. Access paths for files not currently being processed (not open) are normally maintained as if in use (immediately maintained). However, when you create a file, you can specify that the access path should be rebuilt when the file is going to be processed. In this case, the access path for the file is created when a program starts using the file. While this file is active (open), the access path is updated as changes are made to the data. When the file is closed, its access path is invalidated to eliminate constantly maintaining the access path. When the file is opened again, the access path is rebuilt and reflects the current status of the data.

The following is a comparison of immediate and rebuild maintenance.

Immediate	**Rebuild**
– Fast open because the access path is current.	– Slow open because access path must be rebuilt.
– Slower input/output operations because the access path must be updated immediately.	– Faster input/output operations to other files because this access path is not updated immediately.

Note: An access path cannot be created for a file with rebuild maintenance if another concurrently executing user is changing records in the same physical file. Therefore, the file is not opened until the other user has finished changing records. Record retrievals do not interfere with access path creation.

- *When an access path with immediate maintenance should be rebuilt for recovery (RECOVER parameter).*

  If your system goes down abnormally, when you start it again your access paths are rebuilt as part of your data base recovery. Only access paths that have immediate maintenance and that were changing when the system went down are rebuilt. Such a file cannot be used until the access path has been rebuilt. Therefore, it is important that files that are required by most programs or that are needed immediately after CPF startup be rebuilt during CPF startup. You can rebuild the access path either during CPF startup, after CPF startup, or at first use.

The following shows the relationship among duplicate key options, maintenance options, and recovery options.

Duplicate Key Options	**Maintenance Options**	**Recovery Options**
Unique keys	Immediate	Rebuild during CPF startup (*STRCPF) Rebuild after CPF startup (*AFTSTRCPF) Rebuild at first use (*NO,default)
Nonunique keys (FIFO or LIFO)	Immediate	Rebuild during CPF startup (*STRCPF) Rebuild after CPF startup startup (*AFTSTRCPF) Rebuild at first use (*NO,default)
Nonunique keys (FIFO or LIFO)	Rebuild	Rebuild at first use (*NO,default)

If the default recovery option for a file with unique keys is overridden and nonunique keys are entered before the first user open of the file, the access path cannot be rebuilt until the user eliminates the duplicate key records.

- *What disk unit the data (for a physical file) or access paths are on (UNIT parameter).*

  You can request that all members of a file be on the same disk unit. If there is not enough space on the unit for initial allocation, the records are placed on different units, and a message is issued.

- *Whether you want the system to determine how many new and updated records are to be written to the disk unit (FRCRATIO parameter).*

  When records are updated or added to the data base, they are not directly written to the disk unit. You can indicate how many are to be written to the disk unit. This is called forcing records to be written to the data base. If you do not specify forcing, the system determines how many records are written.

Physical File Members

Besides the previously listed information, you can also specify for physical file members the following (parameters are given in parentheses):

- *An expiration date for each member in the file (EXPDATE parameter).* If the expiration date is reached, the system operator is notified when the file is opened. He can then override the expiration date and continue or terminate the job. You can then remove the member if you want to. Each member can have a different expiration date, which is specified when the member is added to the file. (The expiration date can be overridden; see Chapter 8, *Overriding Files.*)

- *The maximum number of records that can be placed in each member (SIZE parameter).* The following formula can be used to determine the maximum:

  $R + (I \bullet N)$

Where	R	is the initial record count
	I	is the number of records (increment) to add each time
	N	is the number of times to add records

  The defaults for R, I, and N are 10000, 1000, and 3, respectively.

  For example, you specify R as 5000, I as 1000, and N as 3. When a member reaches the initial maximum of 5000, the system automatically adds another 1000 to the maximum. 1000 can be added to the initial record count of 5000 three times to make the total maximum 8000. When the total maximum is reached, the operator either terminates or adds an increment and continues.

  Instead of taking the default size or specifying a size, you can specify that there is no maximum size.

- *Whether storage is allocated for members when they are added to the file (ALLOCATE parameter).* The storage allocated would be big enough to contain the initial record count for a member. If you do not allocate storage when the members are added, the storage is allocated as records are written to the member. You can only allocate storage if you specified a maximum size for members.

- *Whether the records for a member are to physically reside together (CONTIG parameter).* If you allocate storage, you can request that the storage be contiguous. That is, all the records in a member are to physically reside together. If there is not enough contiguous storage, noncontiguous allocation is used and an information message is issued to the system operator.

You can use the CRTPF command to create the first member when you create the physical file. Subsequent members must be added using the Add Physical File Member (ADDPFM) command. The following example of adding a member to a physical file uses the CRTPF command used earlier in this chapter in *Creating a Physical File.*

```
CRTPF  FILE(ORDHDRP.DSTPRODLB)
       MBR(*FILE) EXPDATE(*NONE)
       TEXT('Order header physical file')
```

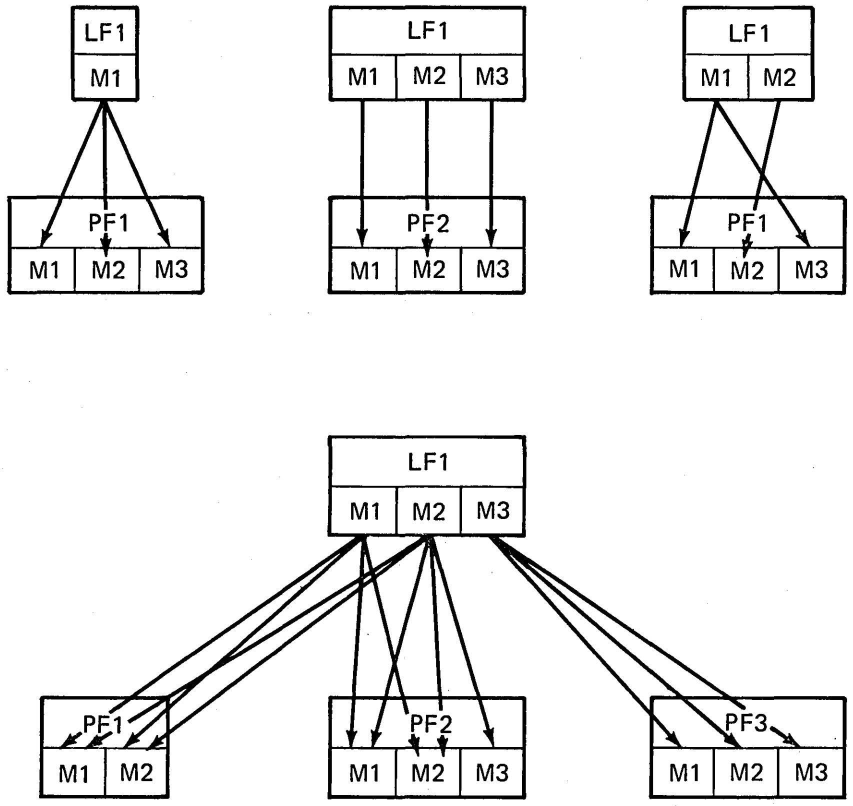
*FILE is the default for the MBR parameter and means that the name of the member is the same as the name of the file. There is no expiration date for the member. The text description of the file is also the text description of the member.

The following example uses the field reference file created previously in this chapter. You do not want a member created. The CRTPF command for the field reference file would be:

```
CRTPF  FILE(DSTREF.DSTPRODLB) SRCFILE(FRSOURCE.QGPL)
       MBR(*NONE) TEXT('Distribution field reference file')
```

Logical File Members

Logical file members allow logical grouping of data through access paths that access different subsets of associated physical file members. One logical file member can be associated with a single physical file member or several physical file members. The following illustrates this concept.



Note that the record formats used with all logical members with a logical file must be defined through DDS when the file is created. If new formats are needed, another logical file must be created. However, if all attributes of an existing logical file's access path are the same, new logical file members can share the access path with an existing logical member in a different logical file.

The attributes of an access path are determined by information specified when the logical file was created and the logical file member was added to the file. This information is specified on the CRTLF and Add Logical File Member (ADDLFM) commands and through the DDS for the file.

When you add a member to a logical file, the access path can be created in four ways:

- Specify no associated physical file members (DTAMBRS parameter) and no access path sharing (no ACCPTHMBR parameter and no ACCPTH keyword). The member is associated with all the physical file members of all physical files in all PFILE keywords specified in the logical file's DDS.

- Specify the associated physical file members (DTAMBRS parameter), but specify no access path sharing (no ACCPTHMBR parameter and no ACCPTH keyword). If you do not specify library names, the library names are obtained from the logical file. If more than one physical file with the same name is specified, you must specify the library names. When more than one physical file member is specified for a physical file, the member names are specified in the order in which records are presented when duplicate key values occur across those members.

- Specify no associated physical file members (DTAMBRS parameter), but do specify access path sharing (ACCPTHMBR parameter and ACCPTH keyword). The data associated with the logical file member is the same as the data associated with the member specified in the ACCPTHMBR parameter.

- Specify the associated physical file members (DTAMBRS parameter) and access path sharing (ACCPTHMBR parameter and ACCPTH keyword).

When you define a record format for a logical file that shares an access path, you can use any fields from the associated physical file record format. The fields do not have to be used in the file that describes the access path. However, all key fields used in the file that describes the access path must be used in the new record format, and the key fields must be specified in the same order.

You can use the CRTLF command to create the first member when you create the logical file. Subsequent members must be added using the Add Logical File Member (ADDLFM) command. The following example of adding a member to a logical file uses the CRTLF command used earlier in this chapter in *Creating a Logical File.*

```
CRTLF  FILE(ORDHDRL.DSTPRODLB)
       MBR(*FILE) DTAMBRS(*ALL)
       TEXT('Order header logical file')
```

*FILE is the default for the MBR parameter and means that the name of the member is the same as the name of the file. All the members of the associated physical file (ORDHDRP) are used in the logical file (ORDHDRL) in the logical file member. The text description of the file is also the text description of the member.

REORGANIZING PHYSICAL FILE MEMBERS

You can reorganize the members in your physical file to:

- Get rid of deleted records to make the space occupied by deleted records available for more records.
- Put the records of a file in the sequence in which you normally access them, thereby minimizing the time required to retrieve records. This is usually not required because System/38 maintains access paths for you.

Members can be reorganized using either of the following:

1. Key fields of the physical file
2. Key fields of a logical file based on the physical file

To reorganize a member, use the Reorganize Physical File Member (RGZPFM) command.

The following RGZPFM command reorganizes a physical file using an access path from a logical file.

```
RGZPFM FILE(ORDHDRP.DSTPRODLB) KEYFILE(ORDFILL.DSTPRODLB)
```

The physical file ORDHDRP has an arrival sequence access path. It has been reorganized using the access path in the logical file ORDFILL. The key field is ORDER. The following illustrates how the records were reordered.

ORDHDRP originally:

CUST	ORDER	ORDATE	. . .
41394	41882	072480	. . .
28674	32133	060280	. . .
56325	38694	062780	. . .

ORDHDRP reorganized using ORDER (ascending sequence):

CUST	ORDER	ORDATE	. . .
28674	32133	060280	. . .
56325	38694	062780	. . .
41394	41882	072480	. . .

Notes:
1. If a file with an arrival sequence access path is reorganized with a keyed access path, the arrival sequence access path is lost.
2. Reorganizing a file compresses deleted records, which changes subsequent relative record numbers.

COMMAND LIST

This is a list of commands related to data base files. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL*.

General

Descriptive Name	Command Name	Function
Delete File	DLTF	Deletes a logical or physical file and its members.

Data Base

Descriptive Name	Command Name	Function
Create Physical File	CRTPF	Creates a physical file optionally using a physical file description (DDS), and, optionally, a member.
Initialize Physical File Member	INZPFM	Initializes a physical file member to the record type specified in the command.
Reorganize Physical File Member	RGZPFM	Reorganizes one or more members in a physical file according to its access path.
Clear Physical File Member	CLRPFM	Removes the data from a specified member of a physical file.
Create Logical File	CRTLF	Creates a logical file from a logical file description (DDS), and, optionally, a member.
Add Physical File Member	ADDPFM	Adds a member to a physical file
Add Logical File Member	ADDLFM	Adds a member to a logical file.
Remove Member	RMVM	Removes a member from a logical or physical file.

Other Commands

This is a list of commands that are also related to data base files but are not part of the functions presented in this chapter.

Descriptive Name	Command Name	Chapter
Copy File	CPYF	Chapter 9, Copying Files
Copy File Interactive	CPYFI	Chapter 9, Copying Files
Display File Description	DSPFD	Chapter 14, Application Documentation
Display File Field Description	DSPFFD	Chapter 14, Application Documentation
Override Data Base File	OVRDBF	Chapter 8, Overriding Files

KEYWORD SUMMARY

This is a list of DDS keywords used for describing physical and logical files.

Keyword	Function	Where Used
ABSVAL	Ignore the sign of the field when sequencing values (use absolute values).	Physical file, key field level • Logical file, key field level.
ACCPTH	Share the access path of a previously created logical or physical file.	Logical file, file level
ALL	Select or omit all records not meeting the select/omit rules.	Logical file, select/omit field level
ALTSEQ	Use an alternate collating sequence for the key.	Physical file, file level Logical file, file level
CHECK	Specifies the following check algorithms that a field value must meet to be valid: • Mandatory enter or fill • Validate name • IBM modulus 10 or 11 self check	Physical file, field level Logical file, field level
CMP	Specifies a comparison or relation, such as equal to, that a field value must meet to be valid or selected.	Physical file, field level Logical file, field level and select/omit field level
COLHDG	Specifies a column heading for a field (used by the Query Utility and other utilities but not by data base management).	Physical file, field level Logical file, field level
CONCAT	Concatenate fields from a physical file into a field in a logical file.	Logical file, field level

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used
DESCEND	Retrieve values in a key field in descending sequence.	Physical file, key field level Logical file, key field level
DIGIT	Fill the zone portion (high-order 4 bits) of each byte of the key field with zeros to build a key value.	Physical file, key field level Logical file, key field level
EDTCDE	Specifies the edit code by which field values are to be displayed.	Physical file, field level Logical file, field level
EDTWRD	Specifies an edit word that describes the form in which field values are to be displayed.	Physical file, field level Logical file, field level
FORMAT	Use a previously described record format.	Physical file, record format level Logical file, record format level
LIFO	Process records with duplicate key values in a last-in-first-out (LIFO) order.	Physical file, file level Logical file, file level
NOALTSEQ	Specifies that no alternate collating sequence is to be used for the key field.	Physical file, key field level Logical file, key field level
PFILE	Specifies the physical files on which the logical file record format is to be based.	Logical file, record format level

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used
RANGE	The field value must be within the limits specified by the range to be valid or selected.	Physical file, field level Logical file, field level and select/omit field level
REF	Retrieve field specifications from the referenced file.	Physical file, record format level and file level
REFFLD	Retrieve field specifications from the referenced field (optionally file).	Physical file, field level
RENAME	Change a physical file field name for use in a logical file record format.	Logical file, field level
SIGNED	Consider the sign of a field when sequencing values.	Physical file, key field level Logical file, key field level
TEXT	Specifies a text description of a new record format or of a field in a new record format.	Physical file, record format level and field level Logical file, record format level and field level

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used
UNIQUE	No duplicate key values allowed.	Physical file, file level Logical file, file level
VALUES	The field value must be one of the values specified to be valid or selected.	Physical file, field level Logical file, field level and select/omit field level
ZONE	Fill the digit portion (low-order 4 bits) of each byte of the key field with zeros to build a key value.	Physical file, key field level Logical file, key field level

Note: The CHECK, EDTCDE, and EDTWRD keywords are not used by the data base but are retrieved when field specifications from the data base are used in a device file.

Chapter 6. Device Support

Before a device can be used on System/38, the system must have at least two types of information. First, a device description must exist for the device; that is, the device must be described to the system. A device description contains information such as device address, device name, device type, model number, and features. Second, a device file must exist for the device. A device file describes data on the device and some of the processing of that data. Any number of device files can be associated with a device, but only one device description can exist for a device.

For some devices, such as remote work stations, corresponding line and control unit descriptions must exist in addition to the device descriptions. See the *Program Product Installation Guide* for more information about line, control unit, and device descriptions.

Note: The device and any associated lines or control units must be varied on. This function is usually performed when the system is started or using the appropriate vary command.

When a device description is created for a device, a device file is created by the system. A system-created device file has the following characteristics:

- The device file name and device name are identical. The name is taken from the Create Device Description (CRTDEVD) command.
- The device file is placed in the system library QSYS.
- The device file is program-described.
- The record format has the same name as the device file and contains one field. The using program describes the field specifications.
- The device file contains default values for device dependent information (for example, for printer files print six lines per inch, and for display files the number of devices associated with the file is one).

The names of the system-created device files are:

- QSYSPRT (printer device file)
- QCARD96 (card device file)
- QDKT (diskette device file)
- QCONSOLE (console device file)

You can create additional device files to fit your needs (such as a card file for jobs needing a special card form).

Device files can be created at any time for:

- Card devices
- Diskette devices
- Printers
- Displays (both display work station and system console)

There are two types of device files: externally described data device files and program described data device files. Both types of files contain file and record format level information. The difference in the two types of files is at the field level. For externally described data device files, the fields are described using DDS (data description specifications). For program described data device files, the fields are described in the program that processes the file; the fields are not described in the file itself. The record is processed as one field.

Both types of device files apply to printers and displays. Only program described data device files are valid for diskette and card devices.

The system-created device files are program described data device files.

PROGRAM DESCRIBED DATA DEVICE FILES

A program described data device file tells the system where the output should go and where the input comes from. Device files have many common attributes, which are specified on the create device file command for the respective devices:

- Spooling information for the output for the file (does not apply to display files):
  - The output queue name
  - The number of copies
  - The maximum number of records that can be spooled for the file
  - The number of file separators
  - When output is scheduled (at job end, at file end, or immediately as it is spooled)
  - Whether the output is to be held on the output queue until the system operator specifically releases it
  - Whether the output is to be saved after it is produced
- Devices on which the file can be used. On the create device file command you can specify that no device is specifically associated with the file. The device can be specified in an override file command (see Chapter 8, *Overriding Files*) or in a change device file command.

- Wait device allocation. The number of seconds the system is to wait for the file resources to be allocated when the file is opened.

- Whether an open data path (ODP) can be shared by different opens of the same file. (An ODP is the path or internal control block through which all I/O operations for a file are performed.) If a file is opened more than once and an ODP is still active for it, the files can share the ODP; a new ODP does not have to be created. Once the shared option is specified for an active ODP for a file, other files can share the ODP.

For each device there are device dependent attributes. Again, these attributes are specified on the create device file commands.

For a card file the device dependent attributes are:

- The hopper (1 or 2) that cards are read from

- The form (card) type

For a diskette file the device dependent attributes are:

- The volume identifiers of the diskettes to be used for the file

- The data file labels on the diskettes

- The location of the diskettes (magazine or slot and starting and ending diskette positions)

- The character code (EBCDIC or ASCII) for the data on the diskettes

- The creation date of an input data file on diskette. If the creation date written on the diskette does not match the date specified in the file description, a message is sent to the system operator, who determines what should be done. The format of the date is the format specified in the system value QDATFMT.

- The expiration date of an output data file on diskette. The expiration date means that the data file cannot be written over until the date has expired. The file is considered to be protected. The format of the date is the format specified in the system value QDATFMT.

The diskette magazine drive can hold two magazines, each containing 10 diskettes, and three slots for manually inserting diskettes.

Before you can use a diskette on the system it must be initialized and have a volume label written on it. When you originally get your diskettes, they are initialized. If a diskette must be reinitialized or its sector size changed to accommodate save/restore, you can use the Initialize Diskette (INZDKT) command.

To use the diskette magazine for other than saving and restoring, the following conventions must be followed (the save/restore considerations are documented in Chapter 19, *Save/Restore*):

- Each volume identifier can be from one to six alphameric characters.
- A volume identifier can apply to more than one volume. That is, you can have multivolume files. (However, you are not required to use the same volume identifier for every volume in a multivolume file.) A multivolume file occupies more than one volume (diskette).
- Within a magazine, the volumes can be in any order, and have no relationship to one another as long as the volumes do not have multivolume files on them.

If you use multivolume files, the following conventions must be followed:

- All volumes must be contained on the same type (one-sided or two-sided) of diskette that contains the first volume of the file.
- All volumes are written to and read from physically sequential diskette locations in the magazine. A multivolume file cannot be split between slots and magazines.
- All volumes must have the same physical record length.
- All volumes must be written in the same character code (EBCDIC or ASCII).
- All volumes after the first volume must be written to diskettes that do not contain active files. (An active file has an expiration date greater than the system date.)
- No volume can be written to a diskette having an extended label area.

For a printer file the device dependent attributes are:

- The type of form to be used for spooled output.
- The form length and width. The system default is 66 lines by 132 positions.
- The number of lines to be printed per inch (either 6 or 8 lines).
- The overflow line. The system default is 60.
- The print image for the file. The system default is the standard system print image QSYSIMAGE.
- The replacement character and whether notification is required for an unprintable character. The replacement character is printed in place of any unprintable character detected. For the 5256 Printer, you can specify a character; for the 5211 or 3262 Printer, a blank is always printed.
- Whether forms alignment is required before the output is printed. If it is required, the system operator is sent a message telling him to align the forms.

- Whether a record is to be folded if it does not fit on one line of the form. If a record is folded, it continues on the next line of the form. If it is not folded, it is truncated at the end of one line.

- Whether a translate table is needed to produce the output. The translate table is used for a character-for-character translation of the user program record data into printable characters. For example, a translate table can be used to translate lowercase letters to uppercase letters. If a translate table is specified, any replacement character that is specified is not used.

For a display file the device dependent attributes are:

- The maximum number of display work stations that can be used with the file

- Whether the device associated with the file can be the requester.

Example of Creating a Program Described Data Device File

In the following example you create a card file, CRD25, for output that is punched on preprinted cards.

```
CRTCRDF  FILE(CRD25.QGPL)
         DEV(*NONE) HOPPER(2)
         OUTQ(PUNCH2) FORMTYPE(CRD25)
         TEXT('Punch file for special card form CRD25')
```

The device associated with this file is defined as being the device specified in your high-level language program, on an Override Card File (OVRCRDF) command, or on a Change Card File (CHGCRDF) command.

The cards used are special preprinted cards and are fed from hopper 2. Output is spooled to the spooling queue PUNCH2.

EXTERNALLY DESCRIBED DATA DEVICE FILES

An externally described data device file for a printer or display includes the same attributes as the program described data device file plus a description of the fields in the records in the file. DDS are used to describe the record format and its fields. Through the record format you can describe file characteristics such as skipping before and after printed lines and field characteristics such as underlining.

For externally described data device files, there is another attribute in the file description besides the attributes listed in *Program Described Data Device Files*. This attribute specifies whether you want the record format checked to determine if the program is using the current record format. Each record format is assigned a level identifier when the file it is associated with is created. When the file is opened, the level identifier can be checked to determine if the record format the program was compiled with is the current record format.

Display Data Description Specifications

See Chapter 7, *Display Device Support* for information about externally described data device files for displays.

Printer Data Description Specifications

Generally, program described data device files are used for printers, and printer control information is specified in the program using the file. However, you can use DDS for a printer file to define fields and control the printing of information through spacing and skipping. One record format can control the printing of more than one line. When you enter the DDS, you can specify which line each field should be printed on as well as where the field should be printed on that line. The data description specifications can specify:

- Indicators to condition the printing of a field
- Edit words and edit codes
- Fields that should be underlined
- Spacing between fields of a record
- Spacing between lines

You can also specify system functions that should be performed when a file is printed. These functions print page numbers, the system date, and the system time.

Field specifications can be retrieved from a previously described field. The previously described field (referenced field) can be in a data base file or can already be defined in the DDS source for the file.

Note: When field specifications are being used from a data base file, binary and packed decimal fields are changed to zoned decimal fields.

The order in which fields are specified for the record format determines the order in which they must be used in the record for a using program. However, the fields need not be specified in the order in which they appear on the printed page.

For complete information about DDS, see the *CPF Reference Manual–DDS*.

Spacing and Skipping Lines

The printing of lines can be controlled by spacing and skipping through an externally described data printer file or through an HLL program (using a program described data printer file). Spacing means to advance one line at a time and skipping means to jump from one line to another.

In the DDS for a printer file, you specify spacing and skipping through keywords:

- SPACEA. Spaces a specified number of lines (one through three) *after* printing one or more lines. SPACEA can be specified at the record format level or the field level. At the record format level, spacing occurs after all lines associated with the record are printed. At the field level, spacing occurs after the line containing the field is printed. Space one line is assumed if you do not use the SPACEA keyword. To specify no spacing, specify a zero as the value for the keyword.

- SPACEB. Spaces a specified number of lines (one through three) *before* printing one or more lines. SPACEB can be specified at the record format level or the field level. At the record format level, spacing occurs before any lines associated with the record are printed. At the field level, spacing occurs before the line containing the field is printed.

- SKIPA. Skips a specified number of lines (one through 112) *after* printing one or more lines. SKIPA can be specified at the file level, record format level, or field level. At the file level, SKIPA must be conditioned with one or more option indicators and skipping occurs after all lines associated with the file are printed. At the record format level, skipping occurs after all lines associated with the record are printed. At the field level, skipping occurs after the line containing the field is printed.

- SKIPB. Skips a specified number of lines (one through 112) *before* printing one or more lines. SKIPB can be specified at the file level, record format level, or field level. At the file level, SKIPB must be conditioned with one or more option indicators and skipping occurs before any lines associated with the file are printed. At the record format level, skipping occurs before all lines associated with the record are printed. At the field level, skipping occurs before the line containing the field is printed.

The SPACEA and SPACEB keywords, and the SKIPA and SKIPB keywords at the record format and field levels can only be specified for records that have no line numbers specified in the DDS.

Name Restrictions

The file name, record format name, and field names should be limited to the number of characters allowed by the high-level language you are using or 10 characters, whichever is smaller; otherwise, another name must be equated to them in the HLL program. Also, these names should follow any other restrictions the high-level languages require.

Field names must be unique within a record format. Record format names must be unique within a file.

Keywords

Keywords on the DDS form let you supply information that is specified less frequently than the information for which specific positions are provided on the form. For example, the keyword UNDERLINE underlines a field on a listing.

A complete list of keywords for printer DDS is contained in the section *Keyword Summary* in this chapter; for display DDS see the *Keyword Summary* in Chapter 7, *Display Device Support.*

Option Indicators

An option indicator passes information from an application program to CPF. CPF tests the indicator before the information associated with the indicator is used. Using option indicators you can condition such things as spacing, skipping, and underlining.

Option indicators can be specified at the file level, record format level, and field level. A decimal value of one for an indicator means that the indicator is on; a decimal value of zero means that the indicator is off. These indicators are one-character fields in the output record but are never printed or displayed.

You can condition a field or keyword using more than one indicator. You can use an AND relationship or an OR relationship. The following illustrates the concept of an AND relationship in printer DDS.

1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A																		
	A										ALLOC	R					17	11	
	A			Ø3		Ø4													SPACE(2)
	A																		
	A																		

Both 03 and 04 must be on before two lines are spaced after printing the record containing the field ALLOC.

An A or blank is specified in position 7 for each additional line required to specify indicators:

```
     A
     A            ALLOC     R          17 11
     A  Ø3 Ø4 Ø6
     AA Ø9                                  SPACE(2)
     A
     A
```

Indicators 03, 04, 06, and 09 must be on before two lines are spaced after printing the record containing the field ALLOC.

You can use as many as nine indicators for each AND relationship; each AND relationship is one *condition.*

The following illustrates the concept of an OR relationship.

```
     A
     A            ALLOC     R          17 11
     A  Ø3
     AO Ø4                                  SPACE(2)
     A
     A
```

The O in position 7 indicates that either 03 or 04 must be on before two lines are spaced after printing the record containing the field ALLOC. Each OR relationship constitutes a new condition. You can use as many as nine conditions (with nine indicators each) for each field or keyword to be selected.

When option indicators are used for field conditioning, the last or only indicator must be on the same line as the field specification. Any keyword on the field specification line cannot be conditioned.

When option indicators are used for keyword conditioning, the last or only indicator must be on the same line as the associated keyword.

If a condition is based on an indicator being off instead of on, you must specify N before the indicator:

```
     A
     A            ALLOC     R          17 11
     A NØ2                                  SPACE(2)
     A
     A
```

Keyword Summary indicates which keywords can use or require indicators.

MISCELLANEOUS FUNCTIONS

Related to device descriptions and files are:

- Print images
- Translate tables
- Edit descriptions

Print Images

A print belt image must be described to System/38 before printing can be performed. When you get your system, the print image for the print belt you ordered is already described to System/38. You get two forms of the print image: one as source in a source file and one as the print image object in the general purpose library QGPL. To change a print image you must change the source and create a new print image.

The source for a print image must contain a header record and input records. Each record must contain 48 or 64 characters (depending on print image size). Only the first 14 characters are used in a header record:

Characters	**Contents**
1 through 5	IMAGE
6	Blank
7 through 10	CHAR or HEX (right-justified)
11	, (comma)
12 through 14	048, 064, 096, 128, or 192 (number of characters in print image)
15 through 48 or 15 through 64	Unused (can be used for comments)

The input records can be entered as either hexadecimal or character. For hexadecimal, the number of input records, according to the size of the character set, is:

Character Set Size	**Number of Records**	**Length of Each Record**
48	2	48
64	2	64
96	4	48
128	4	64
192	6	64

For character, the number of input records, according to the size of the character set, is:

Character Set Size	Number of Records	Length of Each Record
48	1	48
64	1	64
96	2	48
128	2	64
192	3	64

The print image is specified as an attribute of a printer file on the CRTPRTF command.

Translate Tables

A translate table is used for a byte-for-byte translation of data. The tables are used when reading input from a device or producing output for a device, for alternate sequencing, and field translation.

The IBM-supplied translate tables are used for printer output only.

The source for a translate table must be in a source file. For each table, you must enter eight records, each of which must contain 64 characters. The entire table must be 512 characters.

The following shows the source for a translate table that translates lowercase characters to uppercase characters.

```
000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F
404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F
606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F
C0C1C2C3C4C5C6C7C8C98A8B8C8D8E8F90D1D2D3D4D5D6D7D8D99A9B9C9D9E9F
A0A1E2E3E4E5E6E7E8E9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF
C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF
E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
```

- The characters C0 through C9 replace the characters 80 through 89.
- The characters D1 through D9 replace the characters 91 through 99.
- The characters E2 through E9 replace the characters A2 through A9.

The Create Table (CRTTBL) command to create this table looks like this:

```
CRTTBL   TBL(TABLE1) SRCFILE(QTBLSRC) SRCMBR(TABLE1)
         TEXT('Translate lowercase to uppercase')
```

Field Translation

Translate tables can be used for field translation. A field of contiguous characters up to a specified length is translated byte-for-byte. To use the field translate function you must call the translate module QDCXLATE. The CALL command looks like this:

```
CALL PGM(QDCXLATE) PARM(LEN INPUT TBL LIB)
```

Where:

- LEN is a five-digit decimal field containing the length of the field to be translated.
- INPUT is a character field containing the data to be translated.
- TBL is a 10-character field containing the name of the translation table to be used. (The table name must be left-adjusted.)
- LIB is a 10-character field containing the name of the library that contains the translation table. (The library name must be left-adjusted.) If this field is not specified, the library list is used to find the translate table.

When the field has been translated, the input (untranslated) data is replaced with the translated data.

A translate table can be specified as an attribute of a printer file on the CRTPRTF command. In this case, the translate module QDCXLATE is not invoked. Printable characters within the range X'40' through X'FF' are automatically translated.

Edit Descriptions

For System/38 you can define five edit codes, which are called user-defined edit codes in DDS and high-level languages. These edit codes are named using numbers (5, 6, 7, 8, and 9) and can be referenced in DDS or an HLL program by number.

These edit codes are created as a result of creating edit descriptions. An edit description can contain (besides its number):

- Integer mask. Describes the editing of the integer portion of a field. All characters except blank, zero, and ampersand (&) are treated as constants:
  - Blank means to replace the blank with a digit.
  - The leftmost zero means to replace the zero with a digit and terminate zero suppression. All other zeros are treated as constants.
  - Ampersand means to replace the & with a blank.
- Decimal point. Defines what character is used as the decimal point. By default, a period (.) is used.

- Fraction mask. Describes the editing of the fraction portion of a field. Blank and ampersand are the same as for the integer mask. All zeros are treated as constants.

- Fill character. Defines what character is used in each position of a result that is zero-suppressed. By default, a blank is used.

- Floating currency symbol. Defines the floating currency symbol to be used to edit the field.

- Zero balance. Specifies how zero values are to be edited. They can be edited using the fill character or the integer and fraction masks.

- Negative status. Defines the characters that are to follow the edited result of a field if the field is negative.

- Positive status. Defines the characters that are to follow the edited result of a field if the field is positive or zero.

- Left constant. Defines a constant that is to be the leftmost portion of the edited result of a field.

- Right constant. Defines a constant that is to be the rightmost portion of the edited result of a field.

The following are rules you should be aware of when using edit descriptions. These rules are affected by the length and decimal positions of the field being edited.

- The field to be edited is aligned according to the integer and fraction masks.

- The entire integer mask is not always used. The integer mask is truncated immediately to the leftmost digit replace character that could be used, which is based on the number of integers in the field to be edited. Zero suppression termination is remembered from the truncated position.

- The decimal point immediately follows the integer mask and the fraction mask immediately follows the decimal point. If no decimal point is used, the fraction mask immediately follows the integer mask.

- The entire fraction mask is not always used. The fraction mask is truncated immediately to the rightmost digit replace character that could be used, which is based on the number of decimal positions in the field to be edited.

- The width of the edited result is equal to the total of the following:
  - Length of left constant
  - Length of floating currency symbol
  - Length of truncated integer mask
  - Length of decimal point (which is always 1 unless no decimal point is used)
  - Length of truncated fraction mask
  - Length of negative or positive status value
  - Length of right constant

- If either the integer mask or fraction mask does not contain enough digit replace characters for the field to be edited, the field is not edited and is ignored.

- Changing an edit description does not affect previously created record formats.

The file containing the record formats must be created again if the new edit description is to be used.

In the following Create Edit Description (CRTEDTD) command, you are creating an edit description to edit a numeric field and indicate whether the value is a credit or debit.

```
CRTEDTD  EDTD(5) INTMASK('ƀƀƀ,ƀƀƀ,ƀƀ0')
         FRACMASK('ƀƀƀƀ') NEGSTS('DEBITƀ')
         POSSTS('CREDIT') LFTCNS('$') RGTCNS('ƀ**')
```

The field BALNCE contains the value 001234 and has two decimal positions. The edited field looks like this:

```
$ƀƀƀ12.34CREDITƀ**
```

USING PRINTER FILES IN PROGRAMS

To use an externally described data device file in a program, you must specify in the program the name of the file and what records are to be processed. When the application program is compiled, the compiler extracts the file description, and it becomes part of the compiled program. The application program must open and close the file, and issue get (read) and put (write) requests to the file. (Simply stated, an open request connects a file to a program and a close request disconnects a file from a program. See the appropriate HLL reference manual for more information on I/O operations.)

COMMAND LIST

This is a list of commands related to nondisplay device files. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual–CL.*

General

Descriptive Name	Command Name	Function
Delete File	DLTF	Deletes a device file.

Display

Descriptive Name	Command Name	Function
Create Display File	CRTDSPF	Creates a display file.
Change Display File	CHGDSPF	Changes the description of a display file.

Card Device

Descriptive Name	Command Name	Function
Create Card File	CRTCRDF	Creates a card file.
Change Card File	CHGCRDF	Changes the description of a card file.

Printer

Descriptive Name	Command Name	Function
Create Printer File	CRTPRTF	Creates a printer file optionally using a device file description (DDS).
Change Printer File	CHGPRTF	Changes the description of a printer file.

Diskette

Descriptive Name	Command Name	Function
Create Diskette File	CRTDKTF	Creates a diskette file.
Change Diskette File	CHGDKTF	Changes the description of a diskette file.
Delete Diskette Label	DLTDKTLBL	Deletes a file label from a diskette.
Initialize Diskette	INZDKT	Initializes a diskette.
Rename Diskette	RNMDKT	Changes the name of a diskette.
Clear Diskette	CLRDKT	Deletes all files from a diskette and defines a single file called DATA for the entire diskette.
Display Diskette	DSPDKT	Displays the names of the files on a diskette.

Edit Codes

Descriptive Name	Command Name	Function
Create Edit Description	CRTEDTD	Creates a description of a user-defined edit code.
Delete Edit Description	DLTEDTD	Deletes an edit code description.

Print Images

Descriptive Name	Command Name	Function
Create Print Image	CRTPRTIMG	Creates a print image.
Delete Print Image	DLTPRTIMG	Deletes a print image.

Translate Tables

Descriptive Name	Command Name	Function
Create Table	CRTTBL	Creates a table.
Delete Table	DLTTBL	Deletes a table.

Other Commands

This is a list of commands that are also related to files but are not part of the functions presented in this chapter.

Descriptive Name	Command Name	Chapter
Copy File	CPYF	Chapter 9, Copying Files
Display File Description	DSPFD	Chapter 14, Application Documentation
Display File Field Description	DSPFFD	Chapter 14, Application Documentation
Send File	SNDF	Chapter 4, Control Language Programs
Send/Receive File	SNDRCVF	Chapter 4, Control Language Programs
Receive File	RCVF	Chapter 4, Control Language Programs
Cancel Receive	CNLRCV	Chapter 4, Control Language Programs
Wait	WAIT	Chapter 4, Control Language Programs
Override Display File	OVRDSPF	Chapter 8, Overriding Files
Override Card File	OVRCRDF	Chapter 8, Overriding Files
Override Printer File	OVRPRTF	Chapter 8, Overriding Files
Override Diskette File	OVRDKTF	Chapter 8, Overriding Files

KEYWORD SUMMARY

This is a list of DDS keywords used for describing printer files.

Keyword	Function	Where Used	Option Indicator
BLKFOLD	Fold a record at the last blank before the end of the line instead of folding at the actual end of the line. If no blank, break at end of line.	Field level	Optional
DATE	Use the job date as a field in the file and, optionally, edit it according to an edit word.	Field level	
DFT	Initialize a field to a constant value that is the default value for the field.	Field level	
DLTEDT	Do not duplicate edit information when duplicating a field specification.	Field level	
EDTCDE	Specifies the edit code by which field values are to be displayed.	Field level	
EDTWRD	Specifies an edit word that describes the form in which values are to be printed.	Field level	
INDTXT	Describes the use of an option indicator.	All levels	

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
PAGNBR	Print a system-supplied page number in this field (a 4-digit zoned decimal field).	Field Level	Optional
REF	Retrieve field specifications from a referenced data base file.	File Level	
REFFLD	Retrieve field specifications from a data base file (other than the file specified in the REF keyword, if specified).	Field level	
SKIPA	Specifies a line to skip to after printing a line (file level), after printing an entire record (record format level), or after printing the line containing the field associated with this keyword (field level).	All levels	Optional – Field level Record format level Required – File level
SKIPB	Specifies the line to skip to to print the next line of output (file level), to print the lines associated with a record (record format level), or to print the line containing the field associated with this field (field level).	All levels	Optional – Field level Record format level Required – File level

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
SPACEA	Specifies the number of lines to space after printing an entire record (record format level) or to space after printing the line containing the field associated with this keyword (field level).	Field level Record format level	Optional
SPACEB	Specifies the number of lines to space before printing the lines associated with a record (record format level) or before printing the line containing the field associated with this keyword (field level).	Field level Record format level	Optional
TEXT	Specifies a text description of the record format or field.	Field level Record format level	
TIME	Use the system time as a field in the file and, optionally, edit it according to an edit word (default is 0ƀ:ƀƀ:ƀƀ).	Field level (constant field only)	
UNDERLINE	Underline the field.	Field level	Optional

Chapter 7. Display Device Support

Display device files are used to format the display. They describe input and output fields, constants, the use of command function and command attention keys, and the handling of errors.

FORMATTING DISPLAYS

The design of display formats includes describing fields (record formats) and the placement of records on the display device.

Record Formats

The record format is used both for formatting information at a display work station and for passing information to an application program. The record format can contain three types of fields:

- *Input fields.* Fields that are passed from the device to the program when the program reads a record. Input fields can be initialized with a default value (specified in the record format for the device file). If you do not change the field, this data (default value) is passed to the program as input. Input fields that are not initialized are displayed as blanks into which the work station user can enter data. By default, input fields are underlined on the display.

- *Output fields.* Fields that are passed from the program to the device when the program writes a record to a display. Output fields can be provided by the program or from the record format in the device file. Output fields that are provided from the device file are unnamed and are used to display constant information.

- *Output/input fields.* Fields that are passed from the program when the program writes a record to a display and passed to the program when the program reads a record from the display. An output/input field is an output field that is also an input field. Output/input fields are indicated as both fields (B) in DDS. By default, these fields are underlined on the display. Output/input fields are usually used when the program displays data that can be changed by a work station user.

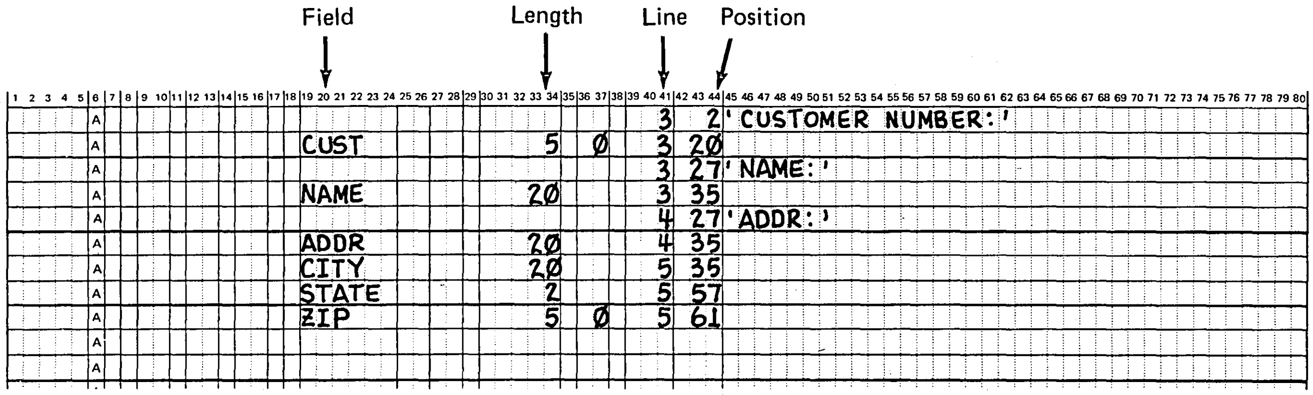
The following record shows output fields and input fields. It was displayed in response to a request (in the form of entering a customer number in an input field) from a work station user.

```
CUST:    41394
ORDER:   41882
ORDATE:  5/24/80
ORDAMT:  $580.00
ARBAL:   $580.00
ENTER NEXT CUSTOMER NUMBER:
```

CUST, ORDER, ORDATE, ORDAMT, ARBAL, and ENTER NEXT CUSTOMER NUMBER are constants. The data associated with these fields (41394, 41882, 5/24/80, $580.00, and $580.00) is displayed in output fields. The data is passed from the application program to the CPF, and CPF displays it. The data associated with the output field ENTER NEXT CUSTOMER NUMBER is an input field. The work station user must enter data into this field (the cursor is positioned at the beginning of the input field).

A record format for a display file describes the format of the record used in the application program and the format of the record when it is displayed (see Figure 16). The fields in the record passed to the program are arranged in the order in which the fields are described in DDS. The order in which the fields are displayed is based on the display positions assigned to the field in the DDS. A field's description contains the field's location on the display, the length of the field, the type of data contained in the field (character or zoned decimal), and the field type (output, input, or output/input).

DDS for Display File:

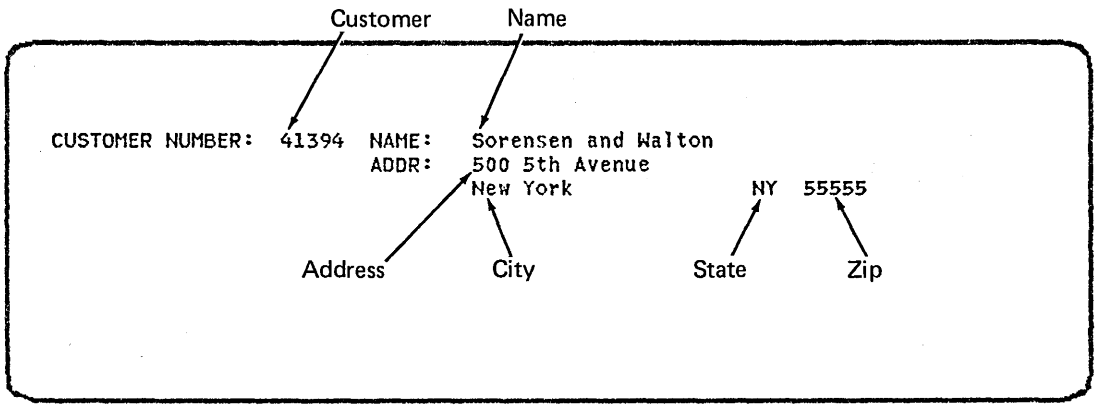

Field
Length
Line
Position

```
     A                                  3  2'CUSTOMER NUMBER:'
     A            CUST           5  Ø   3 2Ø
     A                                  3 27'NAME:'
     A            NAME          2Ø      3 35
     A                                  4 27'ADDR:'
     A            ADDR          2Ø      4 35
     A            CITY          2Ø      5 35
     A            STATE          2      5 57
     A            ZIP            5  Ø   5 61
     A
     A
```

Record Format Used by the Program:

CUST	NAME	ADDR	CITY	STATE	ZIP

Record Format on the Display:



```
CUSTOMER NUMBER:  41394  NAME:   Sorensen and Walton
                         ADDR:   500 5th Avenue
                                 New York              NY   55555
```

Figure 16. Record Formats in the Program and on the Display Device

Location is required for each field except when the field is a hidden field or a message field (H or M specified in usage position in DDS). Line 1, position 1 cannot be entered for location.

The maximum length of a character field is the number of positions remaining (relative to the field's start location) on the display minus one (line 1, position 1 cannot be used). The maximum length of a numeric (zoned decimal) field is 31.

Field specifications can be retrieved from a previously described field. The previously described field (referenced field) can be either in a data base file or already defined in the DDS source for the file. When field specifications are being used from a data base file, binary and packed decimal fields are changed to zoned decimal fields.

Selection of fields can be used to display different data on different output operations instead of defining a different record format for each combination of fields.

Fields in the same record can be defined (in DDS) to overlap each other. That is, two fields could be defined to occupy the same positions on the display. Option indicators can be used to select which of the overlapping fields is to be displayed (see *Indicators*, later in this chapter). If more than one overlapping field is selected, only the first field selected is displayed.

Note: Overlapping cannot be used for subfiles unless the subfile size equals the records per page (see *Subfiles* later in this chapter).

Each field displayed has a beginning attribute character and an ending attribute character associated with it that defines the displayed field. The beginning character precedes the first character of a field and is displayed as a blank. The ending character follows the last character of a field and is displayed as a blank. For example, if you specify a field for positions 2 through 8, the beginning attribute character is in position 1 and the ending attribute character is in position 9. A beginning attribute character can overlay an ending attribute character; that is, they can occupy the same position on the display. These characters are not included in the field length you specify in DDS. No other fields can overlay the beginning attribute character. Therefore, when you design a display you must allow space for each field's beginning attribute character. However, you can use the blank (attribute character) to space between fields when they are displayed.

For complete information about DDS, see the *CPF Reference Manual—DDS*.

Placement of Records

One record format can occupy an entire screen or the screen can be divided to display more than one record format. (Only a beginning attribute character can occupy line 1, position 1.) Figure 17 shows how a screen can be divided. Figure 18 shows invalid divisions of the screen.

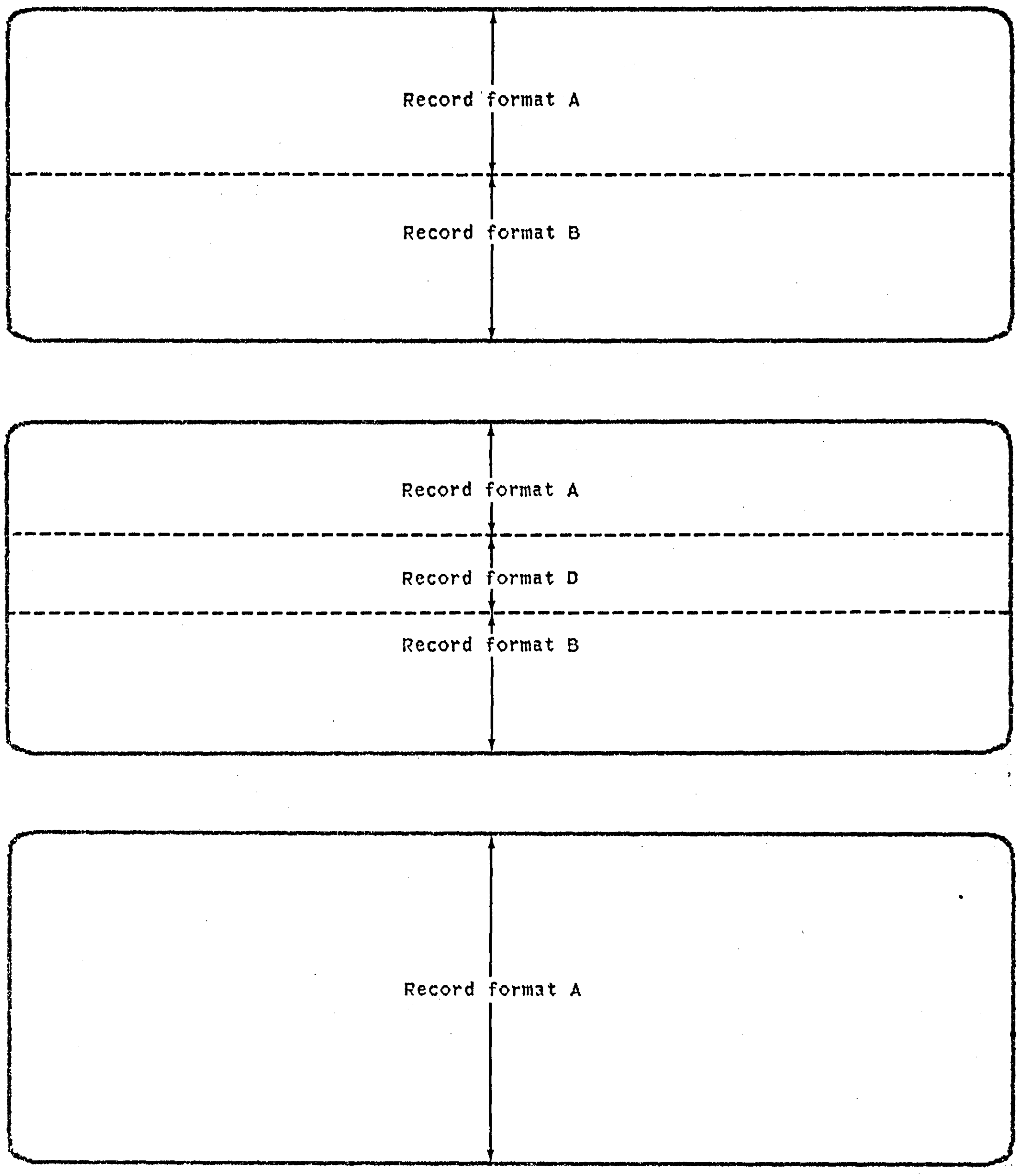


Figure 17. Valid Placement of Records on a Screen

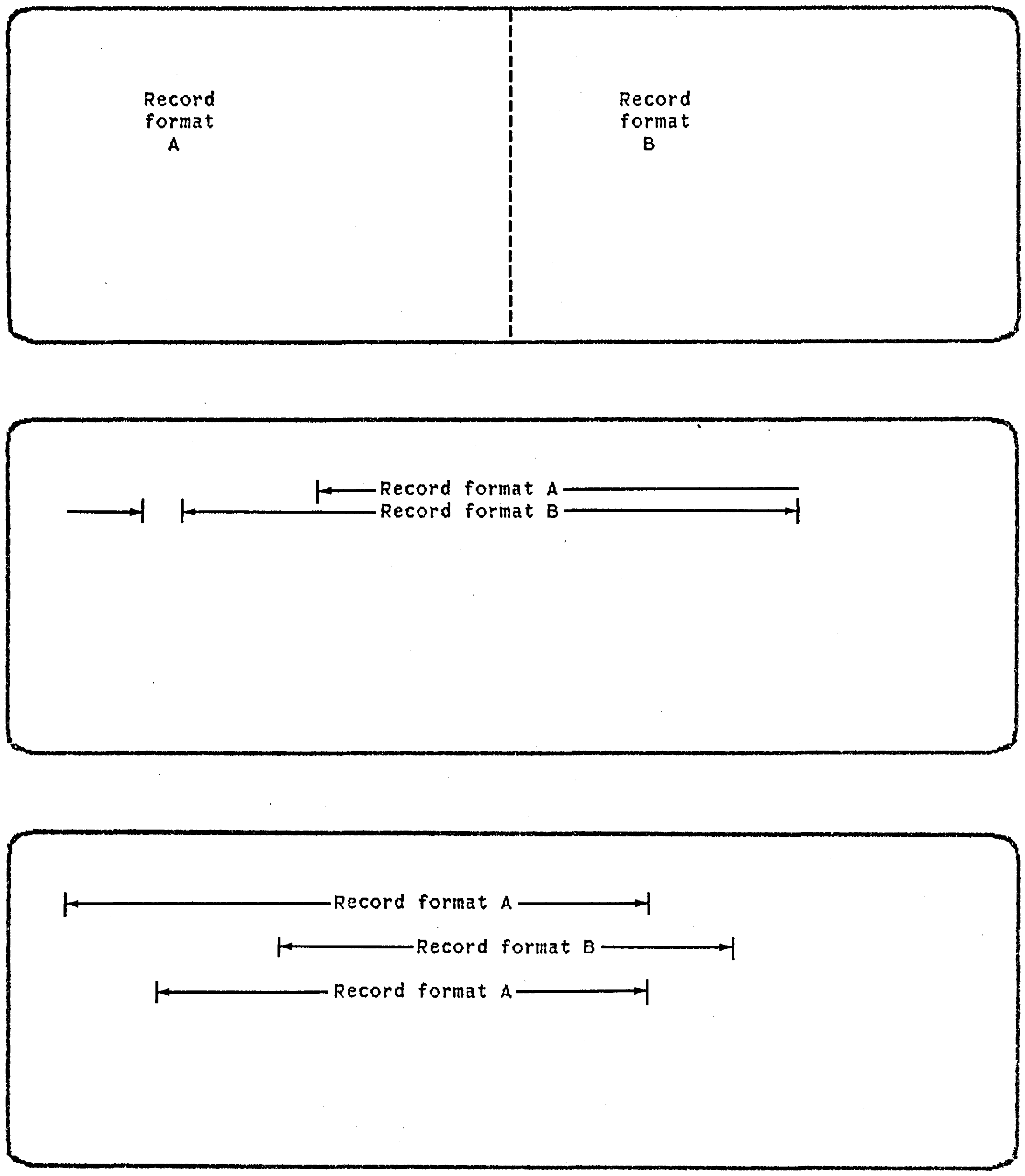


Figure 18. Invalid Placement of Records on a Screen

The formats displayed can change while a file is being processed because information in a display can be erased when new formats are displayed. For example, fields from record format A occupy lines 1 through 4, fields from record format B occupy lines 5 through 7, and fields from record format C occupy lines 8 through 10. Figure 19 shows what happens when record format D, which occupies lines 5 through 9, is displayed when A, B, and C were previously displayed. (The OVERLAY keyword must have been specified for record D; otherwise, record A would be erased also.)

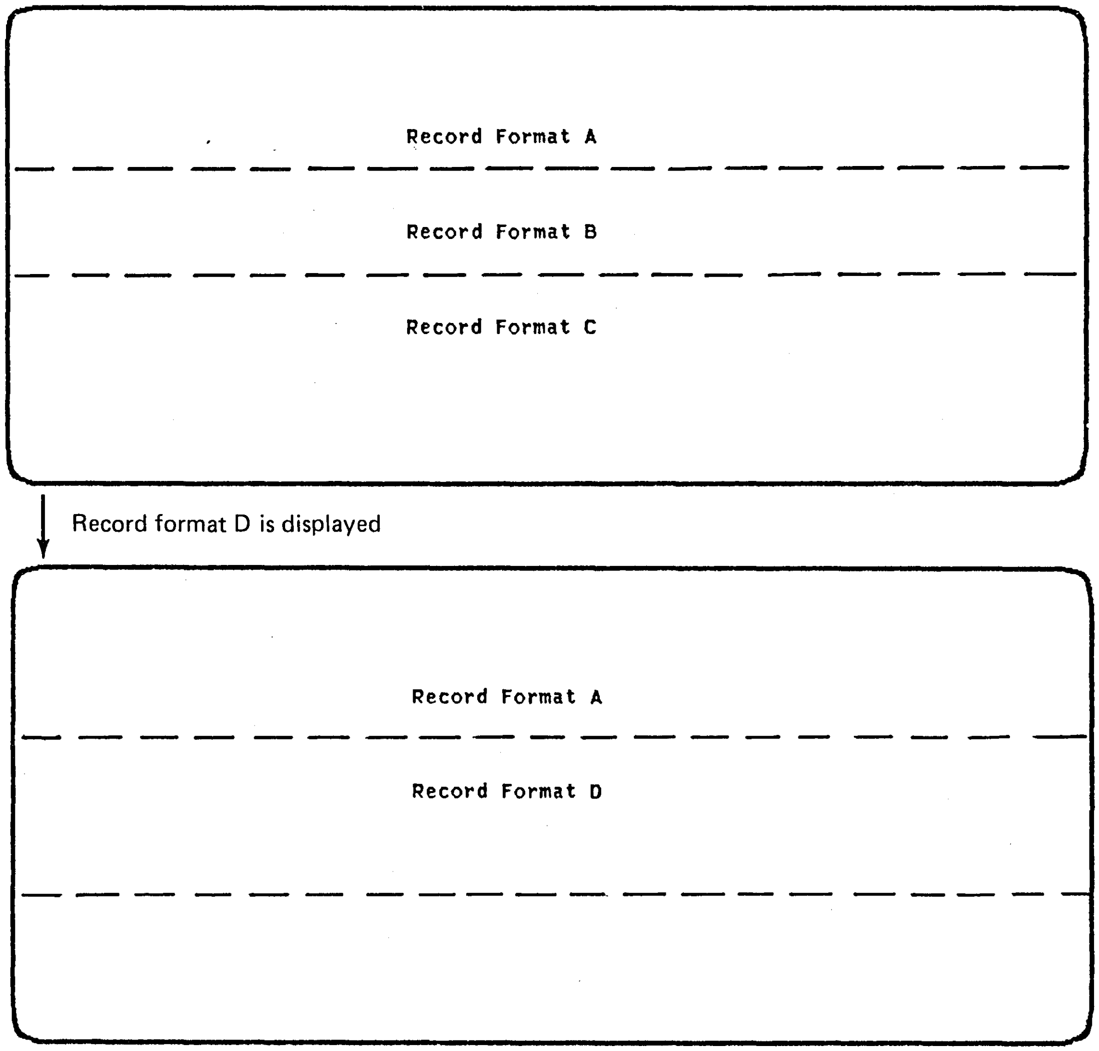


Figure 19. Replacing Record Formats

Records containing input fields should be sent to the display device in the order in which they appear on the display. For example, both records A and B contain input fields and appear on the same display. Record A is displayed on line 3, and record B is displayed on lines 6 and 7. Record A should be sent to the display first. This technique can be used for better performance than if records are sent randomly or in some other order.

DISPLAY ATTRIBUTES

You can emphasize a field of a record on the display by specifying certain display attributes in the DDS for a file. However, any function not supported for your display device is ignored. The display attributes are:

- Underlining a field (DSPATR(UL) keyword), which is the default for input fields.

- Placing vertical separators between the characters in a field (DSPATR(CS) keyword). The DSPATR(CS) keyword is ignored for the system console.

- Highlighting a field by displaying it with greater intensity than is normally used on the display (DSPATR(HI) keyword). The DSPATR(HI) keyword is ignored for the system console.

- Reversing the image of a field from light on dark to dark on light or from dark on light to light on dark (DSPATR(RI) keyword). The DSPATR(RI) keyword is ignored for the system console.

- Placing the cursor at a specific field (DSPATR(PC) keyword).

- Blinking the cursor when a record is displayed (BLINK keyword) or when a field is displayed (DSPATR(BL) keyword). The DSPATR(BL) keyword is ignored for the system console.

- Sounding the audible alarm at the display device (ALARM keyword) when a record is displayed.

VALIDITY CHECKING FUNCTIONS

You can have CPF check the validity of data entered at the display work station. If errors are found, a message is issued to the user so that the user can correct the input before the record is passed to the application program. The validity checking functions are:

- Detecting fields in which at least one character must be entered (CHECK(ME) keyword). Blanks are valid characters. The CHECK(ME) keyword is ignored for the system console.
- Detecting fields in which every position must contain a character (CHECK(MF) keyword). Blanks are valid characters. The CHECK(MF) keyword is ignored for the system console.
- Detecting incorrect data types where character, numeric, or signed numeric data is required.
- Detecting data that is not within the range specified for the field (RANGE keyword).
- Performing comparison checking between data entered and specified constant value (CMP keyword).
- Comparing the data entered to a specific list of valid entries (VALUES keyword).
- Detecting whether a valid name was entered (CHECK(VN) keyword).
- Performing modulus 10 or 11 check digit verification (CHECK(M10) or CHECK(M11) keyword).

Note: If the Dup key is specified (DUP keyword), any validity checking for the field containing the DUP keyword is ignored.

The following shows the valid combinations of validity checking keywords that you can use in DDS (V means valid).

Keyword Specified First	Keywords Subsequently Specified														
	RANGE	VALUES	CMP(EQ)	CMP(NE)	CMP(LT)	CMP(GT)	CMP(LE)	CMP(NL)	CMP(NG)	CMP(GE)	CHECK(ME)	CHECK(MF)	CHECK(VN)	CHECK(M11)	CHECK(M10)
RANGE											V	V		V	V
VALUES											V	V			
CMP(EQ)											V	V			
CMP(NE)											V	V		V	V
CMP(LT)											V	V		V	V
CMP(GT)											V	V		V	V
CMP(LE)											V	V		V	V
CMP(NL)											V	V		V	V
CMP(NG)											V	V		V	V
CMP(GE)											V	V		V	V
CHECK(ME)	V	V	V	V	V	V	V	V	V	V		V	V	V	V
CHECK(MF)	V	V	V	V	V	V	V	V	V	V	V		V	V	V
CHECK(VN)											V	V		V	V
CHECK(M11)	V			V	V	V	V	V	V	V	V	V			
CHECK(M10)	V			V	V	V	V	V	V	V	V	V			

When an error is detected by a validity checking function, the keyboard is locked, the field in error is displayed in reverse image (except on the system console), the cursor is positioned at the beginning of the field in error, and an error message is displayed on the error line.

SCREEN MANAGEMENT FUNCTIONS

When a new record format is displayed for output or to allow input, the existing display is usually erased before the new record format is displayed. If three record formats are on the display at the same time, all three would be erased.

On an output operation you can control certain functions of the display before a new record format is displayed:

- Overlaying a display but not erasing the entire display (OVERLAY keyword). Only records that are completely or partially overlaid are erased; all other records remain on the display.

- Erasing specified records on a display before displaying the next record (ERASE keyword). The ERASE keyword can only be used with the OVERLAY keyword.

- Protecting all input-capable fields on the display not erased before displaying the next record (PROTECT keyword). All input-capable fields are changed to output only fields. The PROTECT keyword can only be used with the OVERLAY keyword. The PROTECT keyword is ignored for the system console.

- Erasing all input and output/input fields on the display not protected before displaying the next record (ERASEIN keyword). (Fields are protected using the DSPATR keyword.) The ERASEIN keyword can only be used with the OVERLAY keyword. For the display work station, if an input field has a modified data tag off or is protected, it is not erased. Also, for the display work station, if the PROTECT keyword is used, ERASEIN is ignored.

- Resetting the modified data tags associated with a displayed record before displaying the next record (MDTOFF keyword). The MDTOFF keyword can only be used with the OVERLAY keyword.

Note: An input-capable field has an attribute that is set on when data is entered into the field. This attribute is called a modified data tag.

- Retaining a record or field on a display (PUTRETAIN keyword) so that it does not have to be retransmitted to the display.

- Locking the keyboard until the last record is displayed so that data cannot be entered into input fields (LOCK keyword).

Also, on an input operation you can control certain functions of the display:

- Initializing a record on the display before reading it (INZRCD keyword). That is, the record does not have to be written to the display before it can be read by the program.
- Unlocking the keyboard so that data can be entered into input fields while the program is processing previously entered input data (UNLOCK keyword). Normally data in all input fields on the display is erased and the modified data tags are reset before the keyboard is unlocked for input.
- Retaining input data on a display (GETRETAIN keyword). The GETRETAIN keyword can only be used with the UNLOCK keyword.
- Setting on a response indicator when data is entered into an input field or data is changed in an output/input field.

SUBFILES

A subfile is a group of records that have the same record format and are read from or written to a display device in one operation. For example, a program reads records from a data base file and writes a subfile of output records. When the entire subfile has been written, the program sends the entire subfile to the display device in one write operation. The work station user enters data in the subfile, then the program reads the subfile and processes each record in the subfile individually, updating the data base file.

A subfile is useful for writing and reading a group of records of the same record format to and from a display. If a subfile is not used, a single record is used; this record contains unique field names for each field on the display. The user program would have to separate the unique combinations of fields into separate records. It is especially useful when the number of records in the group exceeds the number of lines available on the display or where the number of records in the group cannot be determined in advance. For example, a program reads all the line items for an order from a data base file, writes them to a subfile, and displays the first page (group) of line items. The user can scan through the subfile using Roll Up and Roll Down keys and the user can change the records. When the user presses the Enter key, the entire order is read.

You specify what records are to be in a subfile in the DDS for the file. You also specify how many records can be included in one subfile. You can have more than one subfile within the same file. However, only two subfiles can be displayed concurrently.

Records in a subfile can be displayed horizontally and vertically. A horizontally displayed record is complete on one line (see Figure 20). More than one record is displayed on one line. A vertically displayed record is displayed on one or more lines (see Figure 21). Each record begins a new line. Figure 22 shows an example of a vertical subfile and a horizontal subfile being displayed concurrently.

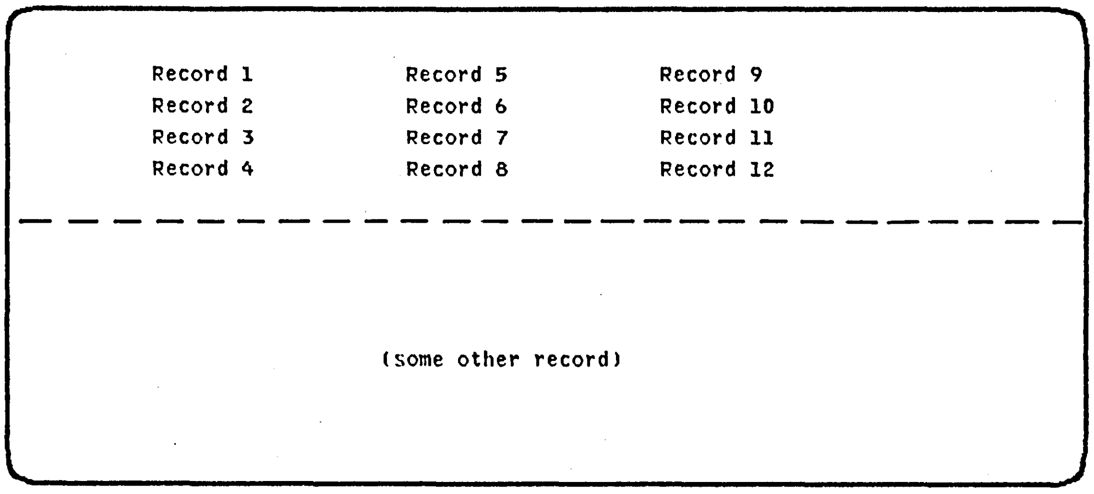


Figure 20. Horizontally Displayed Subfile

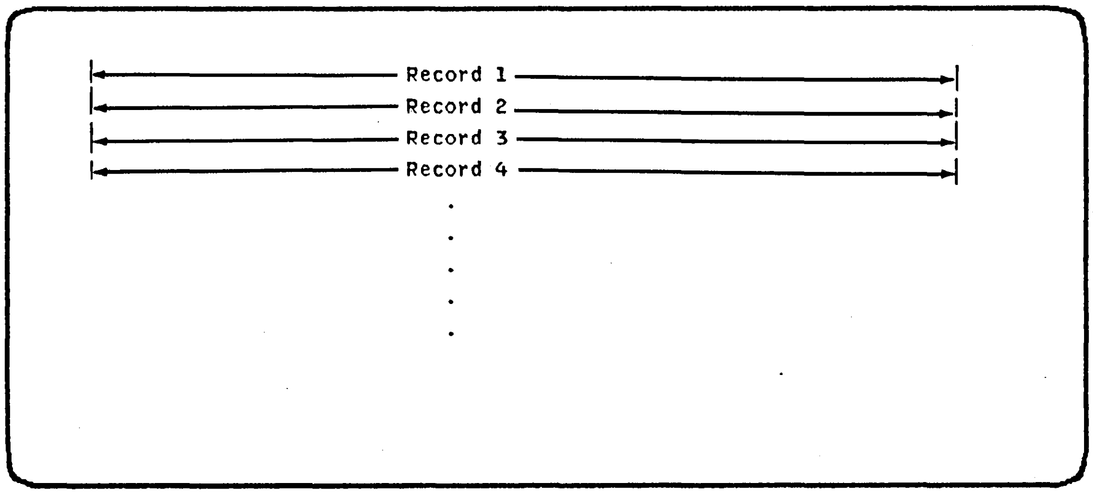


Figure 21. Vertically Displayed Subfile

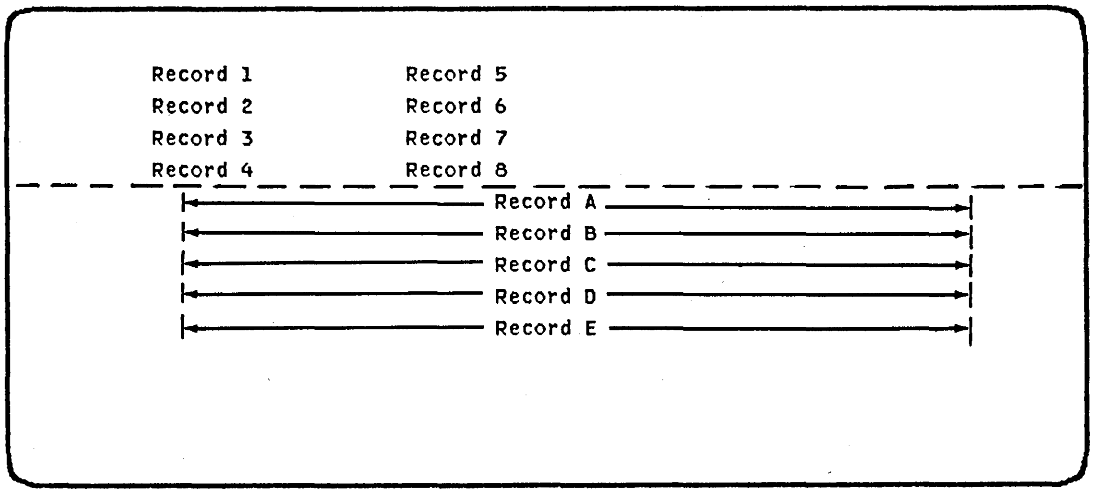


Figure 22. Horizontally and Vertically Displayed Subfiles Displayed Concurrently

If a subfile is larger than the space allowed for the subfile on the screen, the work station user can roll the display from one group of records in the subfile to another. Each group of records displayed concurrently is called a page. When you create a display file with a subfile, you must specify the size of the page for a subfile by specifying the number of records in the page (SFLPAG keyword). Usually page size is based on the number of lines available on the display. You must also specify the size of the subfile by specifying the number of records in the subfile (SFLSIZ keyword). Page size and subfile size can be the same; that is, all records in the subfile fit on one page. When page size equals subfile size, variable length records are supported. One record can take up only a single line while another record can take up more than one display line. Each record is placed in the first record position available in the subfile; this position is always a new line.

You can specify the following for subfiles:

- That the subfile is to be cleared of records before new records are written (SFLCLR keyword). The subfile is not deleted.

- That the subfile is to be deleted (SFLDLT keyword).

- That a command key be used to fold or truncate records in a subfile (SFLDROP keyword). If page size equals subfile size or the subfile is displayed horizontally, SFLDROP is ignored.

- When to begin displaying the records in a subfile (SFLDSP keyword).

- When to display a subfile control record (SFLDSPCTL keyword).

- That the Enter key be used as the Roll Up key and that a command key be used to return to the using program (SFLENTER keyword).

- That a + (plus sign) be displayed in the lower rightmost corner of the subfile display area (page) when there are more records than fit on the display and that the + be replaced by a blank when the last record is displayed (SFLEND keyword).

- The number of spaces between each record on a line when more than one record is displayed on a line (SFLLIN keyword). This is used for a horizontally displayed subfile.

- That you want to roll by a specified number of records instead of by page (SFLROLVAL keyword).

- That a record is to be returned to a program whether or not it was changed by the work station user (SFLNXTCHG keyword).

- That all records in a subfile are to be initialized according to the field descriptions in the device file (SFLINZ keyword).

- That a page of a subfile is to be selected for displaying according to a record number (SFLRCDNBR keyword).

Each subfile needs a subfile record format and a corresponding subfile control record format. The subfile record format defines the fields in each subfile record and performs input (read a subfile record) and output (write new records in the subfile) operations to the subfile.

A subfile control record format controls the normal subfile record that describes the record that is repeated on the display. In addition, heading information can be contained in the subfile control record format (associated with the SFLDSPCTL keyword).

The subfile control record format (indicated by the SFLCTL keyword) requires the SFLSIZ, SFLPAG, and SFLDSP keywords. The following keywords can be used in a subfile control record format but are not required.

- SFLLIN
- SFLEND
- SFLDSPCTL
- SFLCLR
- SFLINZ
- SFLDLT
- SFLDROP
- SFLENTER
- SFLRCDNBR
- SFLROLVAL
- SFLMSG (see *Message Support*, later in this chapter)
- SFLMSGID (see *Message Support*)
- SFLPGMQ (see *Message Support*)

If any input data validity checking is specified for the subfile record, the validity checking is performed before any roll function is performed. If the data fails validity checking, the roll function is not performed.

You can control the positioning of the cursor when a page is displayed.

- The DSPATR(PC) keyword can be used to position the cursor at any field in the first displayed page.
- The DSPATR(PC) keyword can be used to position the cursor at a field of the subfile control record.
- The SFLRCDNBR keyword can be used to position the cursor at the first input field of the specified record to select which page is to be displayed first. The SFLRCDNBR keyword can only be used with the SFLDSP keyword.

- If neither DSPATR(PC) nor SFLRCDNBR is used, the cursor is positioned at the first input field on the display.

MESSAGE SUPPORT

The following are the message handling functions for display support. (DDS keywords are shown in parentheses.)

- Changing the line on which messages are displayed for validity check errors, invalid keys, and user-defined messages (MSGLOC keyword). Initially, the message line is the last line on the display.

- Specifying that messages are contained on a program message queue (SFLMSGRCD and SFLPGMQ keywords). Message keys of messages on the program message queue can be specified to be contained in a field (SFLMSGKEY keyword). Each message is displayed on a separate line and is truncated, if necessary. Position 2 of each line contains an * (asterisk), followed by two blanks, followed by the message (60 characters on the system console, and 76 characters on the display work station, exclusive of the * and the blanks). Second-level text and substitution text are supported.

- Specifying that a message is to be displayed in the standard data management messages method (ERRMSG and ERRMSGID keywords or, for a subfile, SFLMSG and SFLMSGID keywords). The message can be a predefined message in a message file (SFLMSGID keyword) or defined as a constant on the SFLMSG keyword. Substitiution text is not supported. For a message defined as a constant, second-level text is not supported.

See Chapter 12, *Message Handling* for more information.

INDICATORS AND CONDITION NAMES

You can use indicators to pass information from a program to CPF or from CPF to a program. You specify how indicators are to be used through the DDS for the display file.

There are two types of indicators for display files:

- *Option indicators.* Pass information from an application program to CPF.

- *Response indicators.* Pass information from CPF to an application program after the program made an input request. Response indicators can pass information from the work station user to the program by indicating which function keys were pressed by the user or whether data was entered by the user.

Both option and response indicators can be specified at the file level, record format level, and field level. Indicators specified at the file level apply to all record formats within the file. Indicators specified at the record format level or field level are only recognized when the record format name is used in the I/O request. Each indicator occupies a one-character position in the user program record area for the appropriate operation (input response indicators and output option indicators only). A value of 1 for an indicator means that the indicator is on; a value of 0 means that the indicator is off.

See *Option Indicators* in Chapter 6, *Device Support* for how to code option indicators.

Keyword Summary indicates which keywords can use or require option indicators.

In some cases, condition names can be used to select keywords and display locations based on screen size:

- *DS1 (Console, 16 by 64)
- *DS2 (5251, 12 by 80)
- *DS3 (5251, 24 by 80)

Normally, the display files are set up for a 24 by 80 screen. The DSPSIZ keyword specifies which display sizes are valid for a file and indicates which sizes are the primary and secondary screen sizes. (The primary screen size is the first or only DSPSIZ value.)

The screen size condition names let you optimize the use of a single display file for any size screen.

For example, when you are using subfiles, you specify 10 records per page for a 12 by 80 screen and 22 records per page for a 24 by 80 screen:

```
     A
     A  *DS2                                SFLPAG(10)
     A  *DS3                                SFLPAG(22)
     A
     A
```

Note: These condition names cannot be used to reorder fields on the display.

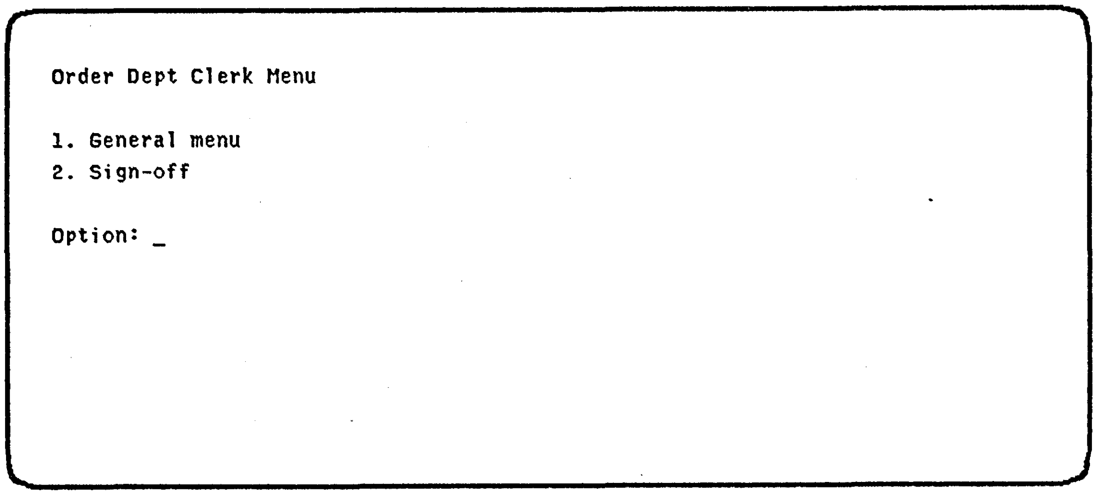
CREATING DISPLAY FILES

A display device file is created using the Create Display File (CRTDSPF) command. For externally described data device files, the DDS for formatting the display is specified on the command. In addition, the CRTDSPF command tells CPF how many and what devices can use the file.

This section contains examples of creating menus, prompts, and information displays.

Formatting a Menu

In this example you describe the record format for the order department clerk menu for your order entry application. Your order department clerk menu looks like this:



```
Order Dept Clerk Menu

1. General menu
2. Sign-off

Option: _
```

Figure 23 shows the DDS to format this menu.

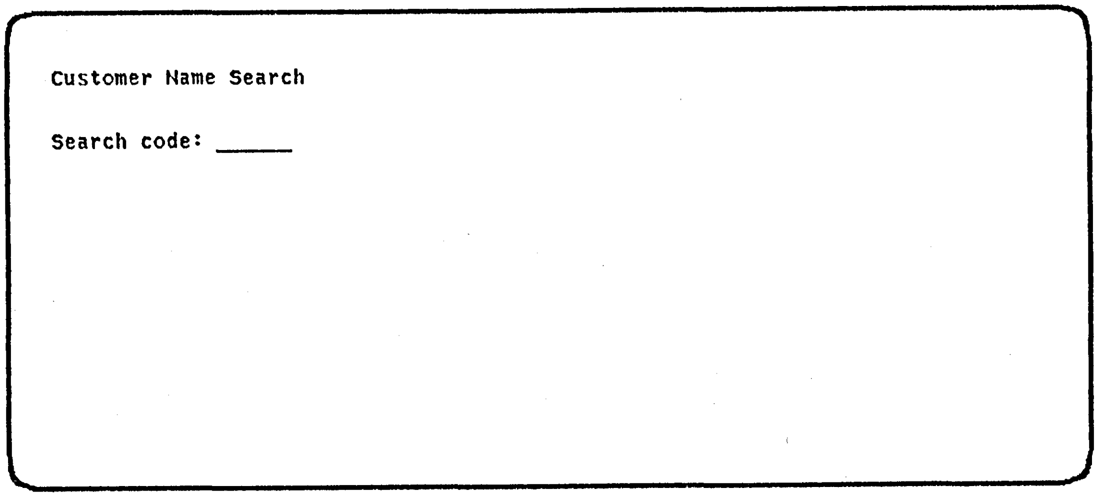
```
     A* OPNØØ5CD ORDER DEPT CLERK MENU
     A
     A          R MENU                      TEXT('Clerk sign-on')
     A                                  1  2'Order Dept Clerk Menu'
     A                                  3  2'1. General menu'
     A                                  4  2'2. Sign-off'
     A                                  6  2'Option:'
     A            RESP           1  ØI  6 1ØRANGE(1 2)
     A                                      CHECK(ME)
     A
     A
```

Figure 23. DDS that Formats a Menu

The valid options are 01 through 02 as indicated by the RANGE keyword for the RESP input field. The CHECK keyword indicates at least one character must be entered in the field.

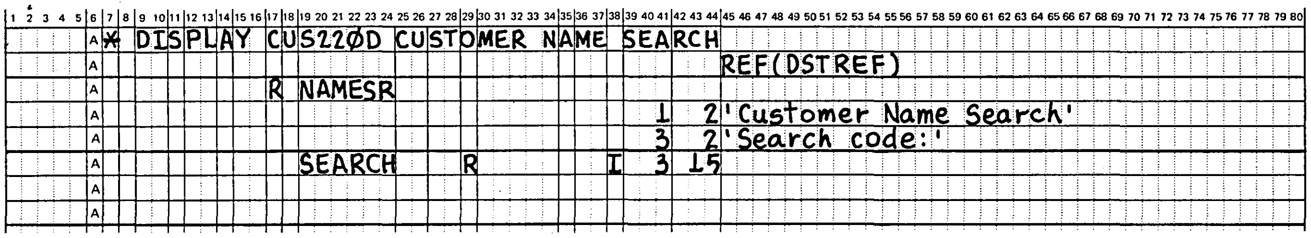
Formatting a Prompt

In this example, you describe the record format for the customer name search prompt. This prompt is used to search the customer name search logical file CUSMSTL1. The search is performed using a zip code. The prompt looks like this:



```
Customer Name Search

Search code: ______
```

Figure 24 shows the DDS to format this prompt.



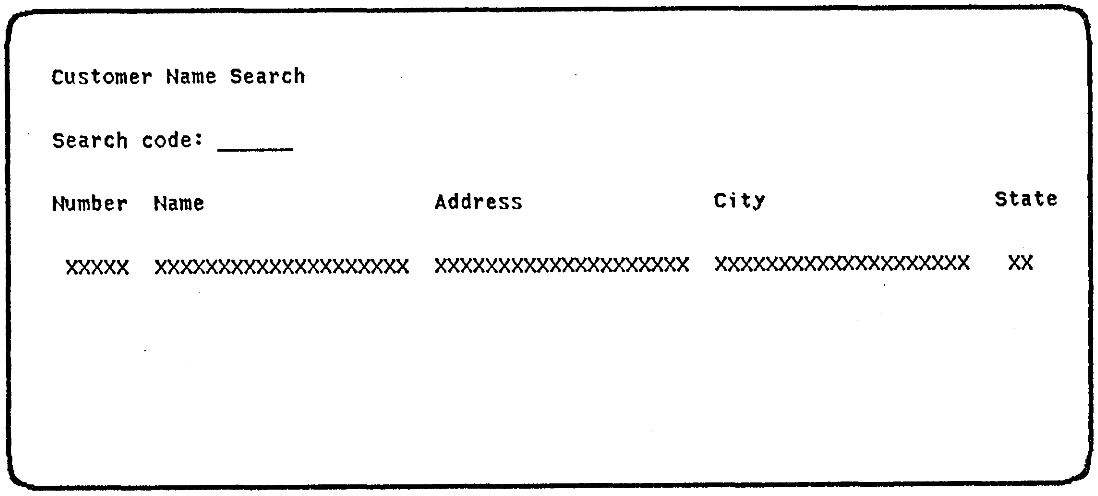
```
     A* DISPLAY CUS220D CUSTOMER NAME SEARCH
     A                                      REF(DSTREF)
     A          R NAMESR
     A                                  1  2'Customer Name Search'
     A                                  3  2'Search code:'
     A            SEARCH    R        I  3 15
     A
     A
```

Figure 24. DDS that Formats a Prompt

The zip code is entered into the search code field, which is underlined by default. The R in position 29 indicates that the field description of SEARCH is contained in another file – the file DSTREF, which is referenced in the REF keyword. By default, the display size is 24 by 80. This prompt cannot be displayed on any other size screen.

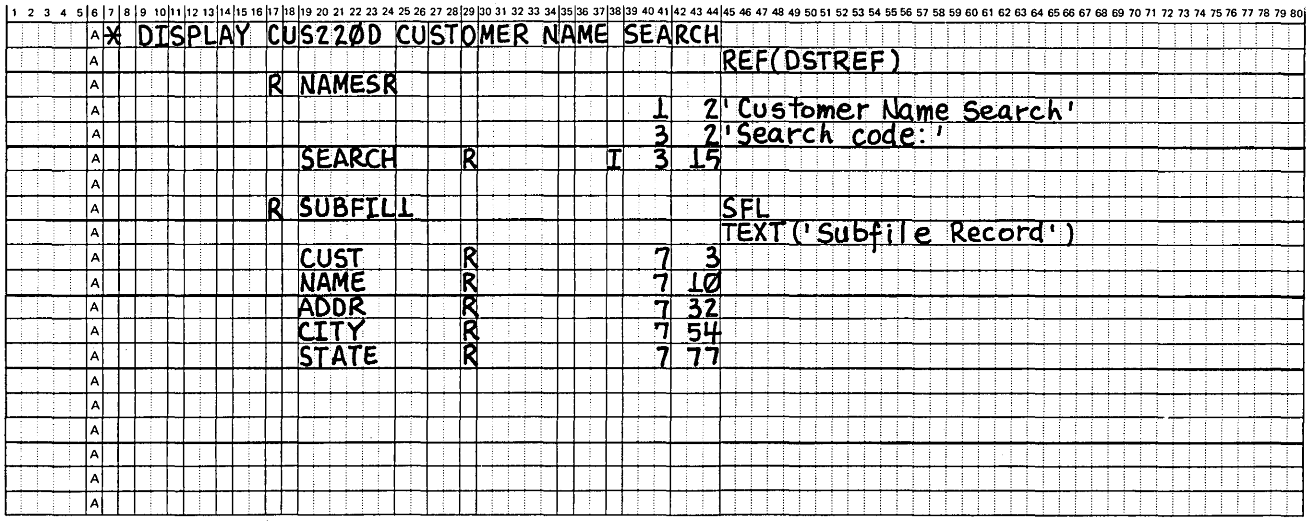
Formatting an Information Display

In this example you describe the record format for the customer name search display, which is displayed as the result of entering a search code. It is displayed on the same display as the search code prompt, which is retained. The records are grouped in a subfile. The complete customer name search display looks like this:



```
Customer Name Search

Search code: ______

Number  Name                  Address               City                  State

 XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX   XX
```

Figure 25 shows the DDS to format this prompt.



```
     A* DISPLAY CUS22ØD CUSTOMER NAME SEARCH
     A                                      REF(DSTREF)
     A          R NAMESR
     A                                  1  2'Customer Name Search'
     A                                  3  2'Search code:'
     A            SEARCH    R        I  3 15
     A
     A          R SUBFIL1                   SFL
     A                                      TEXT('Subfile Record')
     A            CUST      R           7  3
     A            NAME      R           7 1Ø
     A            ADDR      R           7 32
     A            CITY      R           7 54
     A            STATE     R           7 77
     A
     A
     A
     A
     A
     A
```

Figure 25 (Part 1 of 2). DDS that Formats an Information Display

```
     A          R FILCTL1                   SFLCTL(SUBFIL1)
     A N70                                  SFLINZ
     A  70                                  SFLDSPCTL
     A  71                                  SFLDSP
     A                                      SFLSIZ(18)
     A                                      SFLPAG(18)
     A                                      TEXT('Subfile control record')
     A                                      OVERLAY
     A  71                                  ROLLUP(97 'Continue search')
     A                                      CA12(98 'End of program')
     A                                      HELP(99 'Help key')
     A                                      ERRMSG('Rollup to continue search +
     A                                      or key 12 to end program')
     A                                  5  2'Number'
     A                                  5 10'Name'
     A                                  5 32'Address'
     A                                  5 54'City'
     A                                  5 76'State'
     A
     A
```

Figure 25 (Part 2 of 2). DDS that Formats an Information Display

The fields for FILCTL1 are to overlay what is currently on the display (OVERLAY keyword).

The Roll Up key is used to continue searching in the subfile (ROLLUP keyword). The response indicator 97 is set on and passed back to the using program to signal that the Roll Up key is being used. The command attention key 12 sets on indicator 98, which is passed back to the using program to signal that the program is to end. The Help key sets on indicator 99, which is passed back to the using program to signal the help request.

You specify the column headings to be displayed and where (line and position) they are to be displayed.

The record format SUBFIL1 describes the fields in a subfile and the line on which records in the subfile are to begin being displayed. SUBFIL1 is the subfile record fromat (SFL keyword).

The record format FILCTL1 describes the subfile control for the records in SUBFIL1 (SFLCTL keyword). The subfile size and the page size are both 18 records. All records in the subfile appear on one page.

If indicator 70 is not on, the records in the subfile are initialized according to the DDS for the display file. If indicator 70 is on, the subfile control record is displayed. If indicator 71 is on, the records in the subfile are displayed.

Creating a Display File

To actually create a display file, you must enter a Create Display File (CRTDSPF) command. The following CRTDSPF command creates the display file ORD005CD, which contains the order department clerk menu (see *Formatting a Menu,* earlier in this chapter).

```
CRTDSPF  FILE(ORD005CD.DSTPRODLB)
         SRCFILE(QDDSSRC)
         DEV(*REQUESTER)
         TEXT('Order department clerk menu')
```

The DDS source for this file is in the source file QDDSSRC. Because a source member is not specified, the source memeber name is assumed to be the same as ORD005CD.

The device associated with this display file (*REQUESTER) is defined as being the device specified in your high-level language program, in an Override Display File (OVRDSPF) command, or in a Change Display File (CHGDSPF) command.

USING DISPLAY DEVICE FILES IN PROGRAMS

A program communicates with a display device by opening the appropriate display device file to the device. The program can then receive input from and send output to the device. The file should be closed when the program and device have finished communicating. The functions available for communicating with a display device depends on:

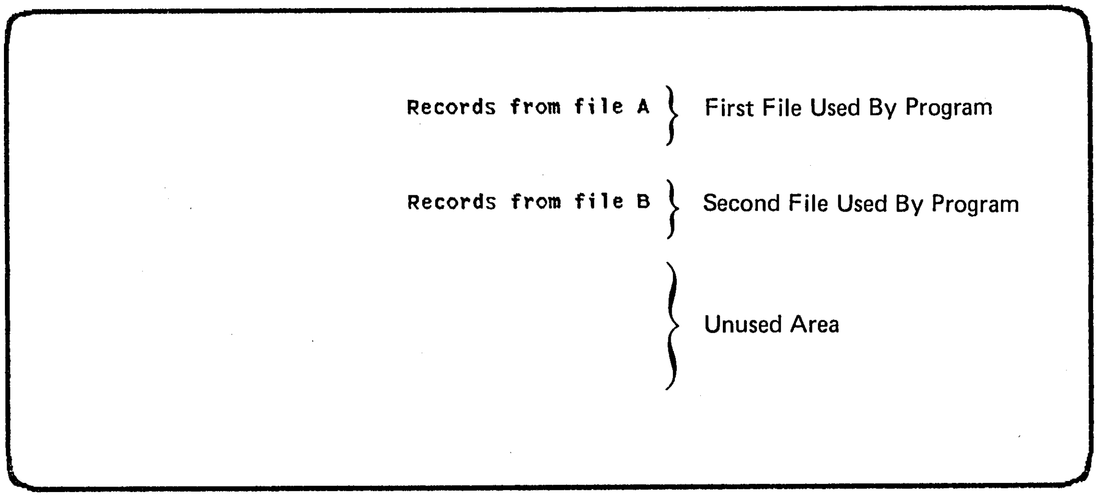
- The kind of display device file used: program described data or externally described data.
- The functions supported by the language in which the program is written.

Note: For control language programs, see Chapter 4, *Control Language Programs*; for other programming languages, see the appropriate HLL reference manual.

CPF provides the device control information necessary for performing input and output operations for the device.

When using a program described data device file to communicate with a display device, only simple display formatting can be performed. A maximum of two display device files can be opened to the same device at the same time within the same program. A file can be opened for input only or output only or both input and output. When a file is opened for either input only or output only, another file can be opened for input only or output only, but it must be the opposite of the other file.

When a program described data device file is opened, the input or output area on the display is treated as a single field on the display. That is, the field length is the same as the record length. The space on the display is assigned as shown in the following. Records for the first file used by the program appear on the first (top) part of the display. Records for the second file appear on the display immediately following the area used by records in the first file.



When using an externally described data device file to communicate with a display device, the display functions that are supported are those specified through DDS. When the application program is compiled, the compiler extracts the file description and it becomes part of the compiled program. During program execution, input and output operations specify record formats that cause:

- The appropriate data to be sent to or received from the device
- The display related functions specified through DDS (for example, validity checking) to be performed.

When a display file is processed, the CPF transforms data from the program to the format specified for the file and displays the data. When data is passed to the program, the data is transformed to the format used by the program.

When you use display files, CPF provides device control information for performing input/output operations with the device. When an input record is requested from the device, CPF issues the request, then removes device control information When you use display files, CPF provides device control information for performing input/output operations with the device. When an input record is requested from the device, CPF issues the request, then removes device control information from the data before passing the data to the application program. In addition, for externally described display files, CPF can pass indicators to the program indicating which fields in the record have been changed by the user.

When your program requests an output operation, it passes the output record to CPF. CPF provides the necessary device control information to display the record. For externally described display files, CPF also adds any constant information specified for the record format when the record is displayed.

COMMAND LIST

This is a list of commands related to display device files. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL.*

General

Descriptive Name	Command Name	Function
Delete File	DLTF	Deletes a device file.

Display

Descriptive Name	Command Name	Function
Create Display File	CRTDSFPF	Creates a display file.
Change Display File	CHGDSPF	Changes the description of a display file.

Other Commands

This is a list of commands that are also related to display files but are not part of the functions presented in this chapter.

Descriptive Name	Command Name	Chapter
Override Display File	OVRDSPF	Chapter 8, Overriding Files
Copy File	CPYF	Chapter 9, Copying Files
Display File Description	DSPFD	Chapter 14, Application Documentation
Display File Field Description	DSPFFD	Chapter 14, Application Documentation
Send File	SNDF	Chapter 4, Control Language Programs
Send/Receive File	SNDRCVF	Chapter 4, Control Language Programs
Receive File	RCVF	Chapter 4, Control Language Programs
Cancel Receive	CNLRCV	Chapter 4, Control Language Programs
Wait	WAIT	Chapter 4, Control Language Programs

KEYWORD SUMMARY

This is a list of DDS keywords used for describing display files.

Keyword	Function	Where Used	Option Indicator
ALARM	An audible alarm is set on when a record is displayed.	Record format level	Optional
ASSUME	Assume that the record is currently on the display when the file is opened.	Record format level	
AUTO	Specifies automatic data functions for fields: • Record advance (RA) • Right adjust with blank fill (RAB) • Right adjust with zero fill (RAZ)	Field level	Optional – record advance only
BLINK	Blink the cursor.	Record format level	Optional

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
BLKFOLD	Fold a field at the last blank before the end of the line instead of folding at the actual end of the line.	Field level	
CAnn	The command key specified by nn is an attention key.	File level Record format level	Optional
CFnn	The command key specified by nn is a function key.	File level Record format level	Optional
CHANGE	Set on a response indicator when data is changed in a field.	Field level Record format level	
CHECK	Specifies the following check algorithms that a field value must meet to be valid: • Mandatory enter (ME) or fill (MF) • Field exit check (FE) • Valid name (VN) • IBM modulus 10 (M10) or 11 (M11) self check	Field level	Optional – mandatory enter only

Keyword	Function	Where Used	Option Indicator
CHGINPDFT	Change the input-capable field to not be underlined, blinked, intensified, and not to have its image reversed.	Field level	
CLEAR	The user program receives control when the Clear key is pressed.	File level Record format level	Optional
CMP	Specifies a comparison, such as equal to, that a field value must meet to be valid.	Field level	
DATE	Use the job date as a field on the display, and, optionally, edit it according to an edit word.	Field level	
DFT	Initialize an input field to a constant value.	Field level	

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
DLTCHK	Ignore (delete) the field validity checking keywords when referencing field specifications from a data base file.	Field level	
DLTEDT	Ignore the edit information when referencing field specifications from a data base file.	Field level	

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
DSPATR	Specifies display attributes for fields: • Column separator (CS) • High intensity (HI) • Underline (UL) • Blink field (BL) • Reverse image (RI) • Protect (PR) • Set modified data tag (MDT) • Nondisplay (ND) • Select by light pen (SP) • Position cursor (PC) • Operator identification (OID)	Field level	Optional – not used for select by light pen or operator identification
DSPSIZ	Specifies the primary display size and any additional display sizes.	File level	
DUP	Allows the the use of the Dup key.	Field level	Optional

Keyword	Function	Where Used	Option Indicator
EDTCDE	Specifies the edit code by which field values are to be displayed.	Field level	
EDTWRD	Specifies an edit word that describes the form in which values are to be displayed.	Field level	
ERASE	Erase one or more records.	Record format level	Optional
ERASEINP	Erase all input fields.	Record format level	Optional
ERRMSG	Specifies the text of the message that should be displayed in the standard data manage-ment messages method.	Field level	Optional
ERRMSGID	Specifies the message identifier of the message that should be displayed in the standard data in the standard data management messages method.	Field level	Optional

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
GETRETAIN	Retain all input data on the display (valid for input only).	Record format level	
HELP	The using program receives control when the Help key is pressed.	File level Record format level	Optional
HOME	The using program receives control when the Home key is pressed.	File level Record format level	Optional
INDTXT	Describes the use of an option indicator.	All levels	
INZRCD	Write a record on the display before it is read.	Record format level	
KEEP	Prevent erasing of the display when a file closes.	Record format level	
LOCK	The keyboard is not to be unlocked.	Record format level	Optional
LOGINP	Copy (log) the input record to the job log.	Record format level	
LOWER	Allow lowercase input.	Field level	

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
LOGOUT	Copy (log) the output record to the job log.	Record format level	Optional
MDTOFF	Reset all modified data tags on the display.	Record format level	Optional
MSGLOC	Specifies what line messages are to be displayed on.	File level	
OPENPRT	The print file is to remain open until the display file is closed.	File level	
OVERLAY	Do not erase the entire display before writing this record.	Record format level	Optional
PASSRCD	Specifies the record format to be used when unformatted data is passed.	File level	
PRINT	The Print key can be used to print the display.	File level	Optional

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
PROTECT	Protect all input-capable fields not erased on an overlay.	Record format level	Optional
PUTRETAIN	Retain a record (at the record format level) or field (at the field level) on the display.	Field level Record format level	Optional
RANGE	The field value must be within the limits specified in the entry position.	Field level	
REF	Field specifications for a device file are to be retrieved from a data base file.	File level	
REFFLD	Field specifications for a field are to be retrieved from a data base file (other than the file specified in the REF keyword, if specified).	Field level	

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
ROLLDOWN	The using program receives control when the Roll Down key is pressed.	File level Record format level	Optional
ROLLUP	The using program receives control when the Roll Up key is pressed.	File level Record format level	Optional
RTGAID	Place the AID byte (command key indication) into the routing data as a 2-byte identifier.	Record format level	
RTGCON	Place the specified literal into the routing data at a specified location.	Record format level	
RTGDEV	Place the 10-character device name into the routing data.	Record format level	
RTGDEVCLS	Place the 10-character device name into the routing data.	Record format level	

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
RTGFIRST	Specifies that the first field received be placed in the routing data.	Record format level	
RTGFLD	Specifies that the field data be placed in the routing data.	Field level	
RTGFMT	Place the format name into the routing data.	Record format level	
RTGPOS	Place data received from a device into the routing data.	Record format level	
SETOFF	Set off a response indicator.	Record format level	
SFL	The record is a subfile.	Record format level	
SFLCLR	Clear the subfile of data but do not delete the subfile.	Record format level	Required

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
SFLCTL	The record is a subfile control record.	Record format level	
SFLDLT	Delete the subfile.	Record format level	Required
SFLDROP	Assign a command key to be used to fold or truncate records of a subfile.	Record format level	
SFLDSP	Specifies when to begin displaying subfile records.	Record format level	Optional
SFLDSPCTL	Specifies when to display a subfile control record.	Record format level	Optional
SFLEND	Display a + (plus sign) when there are more records than can fit on a display page and replace a + with a blank when the last record in the subfile is on the display.	Record format level	Required

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
SFLENTER	The Enter key is used as the Roll Up key, and a command key is used to return to the using program.	Record format level	
SFLINZ	Initialize all records within the subfile.	Record format level	Optional
SFLLIN	Specifies the number of spaces between each record on a line. This number is used to calculate the subsequent field positions for records displayed on the same line.	Record format level	
SFLMSG	Specifies the text of the message that should be displayed in the standard data management messages method. Used for subfiles.	Record format level	Optional

Keyword	Function	Where Used	Option Indicator
SFLMSGID	Specifies the message identifier of the message that should be displayed in the standard data management messages method. Used for subfiles.	Record format level	Optional
SFLMSGKEY	The specified field contains a message key of a message on a program message queue.	Field level	
SFLMSGRCD	Specifies that the subfile contains messages from a program message queue.	Record format level	
SFLNXTCHG	Return a record to a using program whether or not the fields in the record format were changed by the work station user.	Record format level	Optional

KEYWORD SUMMARY (continued)

Keyword	Function	Where Used	Option Indicator
SFLPGMQ	Specifies the name of the program message queue for displaying messages.	Field level	
SFLPAG	Specifies how many records can be displayed in a subfile at one time.	Record format level	
SFLRCDNBR	Display a page of a subfile according to a record number.	Field level	
SFLROLVAL	The specified field and appropriate roll key are used to roll the subfile records that are displayed.	Record format level	
SFLSIZE	Specifies the number of records that can be contained in a subfile storage area.	Record format level	
TEXT	Specifies a text description for a record format or field.	Field level Record format level	

Keyword	Function	Where Used	Option Indicator
TIME	Use the system time as a field on the display and, optionally, edit it according to an edit word or edit code (default is 0ƀ:ƀƀ:ƀƀ).	Field level	
UNLOCK	Unlock the keyboard so the next record can be entered.	Record format level	
USRDFN	The data is a user-defined data stream.	Record format level	
VALUES	The field value must be one of the values specified in the entry position.	Field level	
VLDCMDKEY	Set a response indicator on if a valid command key (a key associated with a keyword) is pressed to return control to the program.	File level Record format level	

Chapter 8. Overriding Files

When you create an application program, files are associated with it by the file names specified in the program. System/38 lets you override these file names and the attributes of the specified file when you compile a program or execute a program. Because files are defined outside an application program, the program can be written so that it is independent of which file is being used.

Overriding a file is different from changing a file in that an override does not permanently change the attributes of a file or which file is used in a program. For example, if you override the number of copies for a print file by requesting six copies instead of two, the file description for the print file would still specify two copies, but six copies are printed. A file override lasts either as long as the program that contains the override command is executing or, if the override command was entered interactively, as long as the routing step is executing. (See Chapter 16, *Work Management* for information about routing steps.)

File override commands can be entered as part of a batch job, as a command in an interactive job, and as part of a control language program.

OVERRIDING FILE ATTRIBUTES

The simplest form of overriding a file is to override some attributes of the file. File attributes are established as a result of the following:

- Create file and add member commands. Initially, these commands establish the file attributes.

- Program using the files. At compile time, the using program can override (change or add to) the file attributes.

- Override commands. At program execution time, these commands can override the file attributes previously established by the create commands and the using program.

For example, you created a printer file OUTPUT whose attributes are:

- Form size of 66 by 132

- Six lines per inch

- Two copies of printed output

- Two pages for file separators

The Create Printer File (CRTPRTF) command looked like this:

```
CRTPRTF  FILE(OUTPUT.QGPL) DEV(PRINTER)
         FORMSIZE(66 132) LPI(6)
         COPIES(2) FILESEP(2)
```

This printer file is referenced in your application program. However, before you run the application program you want to change the number of printed copies of output to three. The override command looks like this:

```
OVRPRTF FILE(OUTPUT) COPIES(3)
```

Then you call the application program, and three copies of the output are printed.

OVERRIDING WHICH FILE IS USED IN A PROGRAM

Another simple form of overriding a file is to change which file is used in a program. For example, you want the output from your application program to be printed using the printer file REPORTS instead of the printer file OUTPUT (OUTPUT is specified in the application program). Before you run the program you enter the following command:

```
OVRPRTF FILE(OUTPUT) TOFILE(REPORTS)
```

Note: The file REPORTS must have been created by a CRTPRTF command before it can be used.

APPLYING OVERRIDES

When you have more than one override for the same file in a series of programs (nested programs), there is a certain order in which the overrides are applied to the file. For example:

```
OVRPRTF  FILE(OUTPUT)  COPIES(6)  SPOOL(*YES)
CALL  PGM(A)

                    Program A

                    OVRPRTF  FILE(OUTPUT)  COPIES(2)  LPI(6)
                    CALL  PGM(X)
```

When program X opens the file OUTPUT, the following overrides are used:

- COPIES(6)
- SPOOL(*YES)
- LPI(6)

The override COPIES(2) is not used because it was overridden by COPIES(6). The first override command (issued before the call to program A) overrides the second override command (embedded in program A). If there had been a third override for OUTPUT embedded in program X, it would have been overridden by the second and first overrides, respectively.

A similar situation exists when you change the name of the file to be used in the program and you also override some of the attributes of the file. For example:

```
OVRDBF  FILE(PAYROLL)  MBR(CURRENT)
CALL  PROG1

                  Program PROG1
                  OVRDBF  FILE(INPUT)  TOFILE(PAYROLL)
                  CALL  PROG2
```

When program PROG2 goes to open INPUT, it opens PAYROLL instead. Also, the member used for processing is CURRENT. The file override that changes the file used by the program is applied first. Then, all subsequent file overrides in the same or prior invocations of programs referencing the file are applied (inverse invocation order).

DELETING OVERRIDES

You can delete active file overrides that no longer apply to your job. You can delete a specified override or all overrides. To identify an override, use the file name specified in the FILE parameter of the override command. For example,

```
OVRPRTF  FILE(OUTPUT)  COPIES(6)  SPOOL(*YES)
CALL  PGM(A)

                  Program A
                  OVRPRTF  FILE(OUTPUT)  COPIES(2)  LPI(6)
                  CALL  PGM(X)

DLTOVR  FILE(OUTPUT)
```

The Delete Override (DLTOVR) command deletes the overrides for file OUTPUT before another program is called that uses OUTPUT.

DATA BASE FILE CONSIDERATIONS

For a data base file, you can change (override) some attributes that are not present in the file description (from the create file command) but were specified in the program using the file. These attributes are:

- Member name. The member from the file you want to use in the program. If not specified, the first (oldest) member in the file is used.

- Position. The position in the file at which processing is to begin. Your options are:
  - At the beginning of the file (default)
  - At the end of the file
  - At a specified record, using a relative record number
  - At a specified record, using a specific record format and key field value

- Record format lock. The lock state for a specified record format. Your options are:
  - Shared for read
  - Shared for update
  - Shared no update
  - Exclusive allow read
  - Exclusive

  (See *Allocating Resources* in Chapter 16, *Work Management* for an explanation of the lock states.)

- Expiration check. The expiration date of the member can be checked to determine if the date has expired. If the date has expired, the override is not applied and the system sends you a message. You can then have the override applied or not. If you do not have the date checked but the date has expired, the system sends you a warning message but the override is applied.

- Inhibit write. You can inhibit (stop) insertions and updates to a file or deletion from a file. Inhibit write is useful when using production libraries in test mode.

SHARED FILES

The SHARE parameter on the override commands allows an open data path (ODP) to be shared between two or more programs executing in the same routing step. If not specified otherwise, every time a file is opened a new ODP is created. (An ODP is the path through which all I/O operations for the file are performed.) You can specify on an override command that, if a file is opened more than once and an ODP is still active for it, the active ODP for the file can be used with the current open of the file and a new ODP does not have to be created. SHARE(*YES) must be specified for the first open and other opens of the same file for the ODP to be shared.

By default, data base allows one file to be concurrently accessed and changed by many users. The SHARE parameter allows even closer sharing in a job such as sharing the file, its status, its position, and its buffer. Two programs using a shared file can pass the file back and forth automatically.

SECURING FILES

When you grant other users the authority to execute a program using your (the owner's) user profile, you may want to protect your files from unwanted overrides. In addition, when you have several levels of nested programs, you may want to protect files from unwanted overrides.

You can prevent file overrides for a program by specifying the SECURE(*YES) parameter on the file override command in the program for each file needing protection. This protects your files from subsequent overrides and prevents the use of files not specified in the program.

The following shows an example of a protected file.

```
OVRPRTF FILE(PRINT1) SPOOL(*NO)
OVRDBF FILE(NEWEMP) TOFILE(OLDEMP) MBR(N67)
CALL PGM(CHECK)
  •
  • (CHECK program)
  •
  OVRDBF FILE(INPUT) TOFILE(NEWEMP) MBR(N77) SECURE(*YES)
  •
  •
  •
  CALL PGM(NEMPRPT)
    •
    • (NEMPRPT program)
    • (INPUT and PRINT1 are opened)
  DLTOVR FILE(*ALL)
  CALL PGM(NEMPLST)
    • (NEMPLST program)
    • (NEWEMP and PRINT1 are opened)
    •
```

Because the OVRDBF command within the program CHECK specifies SECURE(*YES), the first OVRDBF command referencing NEWEMP is not applied. Therefore, OLDEMP is not used in the program CHECK. The OVRPRTF command is applied for the file PRINT1 every time PRINT1 is opened. The DLTOVR command within the program CHECK deletes only the overrides specified in the program CHECK.

FILE REDIRECTION

File redirection means that you changed the file or the type of file to be processed in a program when you overrode a file. For example, you change from one data base file to another data base file or change from a card input file to a display file.

When you change the file that is used in a program but do not change the type of file, the file changed to is processed in the same manner as the original file would have been as long as the record format the program was compiled with matches the file changed to. If you change to a different type of file, the device dependent characteristics of the file are ignored and certain defaults taken. The following summarizes the defaults taken and what is ignored.

From	**To**
Printer	Card: The first 96 characters of the record are punched and printed on cards. Printer control information is ignored.
	Diskette: The first 128 characters of each record are written on diskette. If a two-sided diskette allows 256-byte records, the first 256 characters are written on diskette. Diskette label information must be provided on the override command. Printer control information is ignored.
	Display: Records are written to the display with each record overlaying the previous record. You request each record with the Enter key. Printer control information is ignored.
	Data base: Records are written to the data base in sequential order. Printer control information is ignored.
Card input	Data base: Records are retrieved from the data base. One record is read as a single field. Card control information is ignored.
	Display: Records are retrieved from the display one at a time. Card control information is ignored.

From	To
Card output	Printer: Punch records are printed. Card control information is ignored.
	Diskette: Records are written on diskette. Diskette label information must be provided on the override command. Card control information is ignored.
	Display: Punch records are written to the display with each record overlaying the previous record. You request each output record with the Enter key. Card control information is ignored.
	Data base: Records are written to the data base in sequential order. Card control information is ignored.
Diskette input	Card: Only records of at the most 96 characters are transferred to the program. Diskette label information is ignored.
	Data base: Records are retrieved in sequential order. One record is read as a single field. Diskette label information is ignored.
	Display: Records are retrieved from the display one at a time. A nonfield-level device file must be specified. Diskette label information is ignored.
Diskette output	Printer: Records are printed.
	Card: The first 96 character of each record are punched and printed.
	Data base: Records are written to the data base in sequential order.
	Display: Records are written to the display with each record overlaying the previous record. You request each output record with the Enter key.

From	To
Display input	Card: Up to 96 characters of the input record are read. If less than 96 characters are read, the record is padded with blanks until 96 characters are reached.
	Diskette: Up to 128 character of the input record are read. If less than 128 characters are read, the record is padded with blanks until 128 characters are reached. If a two-sided diskette allows 256-byte records, 256 characters are read. 256-byte records are also padded with blanks. Diskette label information must be provided on the override command.
	Data base: Input records are retrieved.
Data base input (sequentially processed)	Card: Only records of at the most 96 characters are transmitted to a program.
	Display: Records are retrieved from the display one at a time. A nonfield-level device file must be specified.
	Diskette: Records are retrieved from diskettes. Diskette label information must be provided on the override command.
Data base output (sequentially processed)	Card: Only the first 96 characters are punched and printed.
	Printer: The first 132 characters are printed when no folding is specified. If folding isspecified, all of a record is printed.
	Diskette: The first 128 characters are written on diskette. If a two-sided diskette allows 256-byte records, 56 characters are written on diskette. Diskette label information must be specified on the override command.
	Display: Records are written to the display with each record overlaying the previous record. You can request each output record using the Enter key.

The following chart summarizes valid file redirections. I means input file; O means output file; O/I means output/input file.

To	From File				
File	Printer	Card	Diskette	Display	Data Base
Printer	O*	O	O		O
Card	O	O	O	O	O
		I	I	I	I
Diskette	O	O	O	O	O
		I	I	I	I
Display	O	O	O	O/I	O
		I	I		I
Data Base	O	O	O	O	O
		I	I	I	I

* Redirected to a different type of printer.

There is no redirection for card combined files or nonsequentially processed data base files other than to a file of the same type.

COMMAND LIST

This is a list of commands related to overriding files. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL.*

Descriptive Name	Command Name	Function
Override Card File	OVRCRDF	Overrides a card file.
Override Data Base File	OVRDBF	Overrides a data base file.
Override Diskette File	OVRDKTF	Overrides a diskette file.
Override Display File	OVRDSPF	Overrides a display file.
Override Printer File	OVRPRTF	Overrides a print file.
Delete Override	DLTOVR	Deletes previous override commands.
Display Override	DSPOVR	Displays active override information.

Chapter 9. Copying Files

On System/38 you can copy both data base files and device files. As you copy files you can copy the whole file; or you can copy only a portion of a file by selecting a subset of the records.

You can copy an entire file or only part of a file. Besides specifying members to be copied, you can select the parts of a file to be copied by specifying:

- From and to record numbers
- From and to record keys
- A character value to be present in a field or record
- A value to be present in a field
- A record format
- A number of records
- That fields be dropped from the file being copied

You can copy from:

- A data base file to a data base file
- A device file to a data base file
- A data base file to a device file
- A device file to a device file

The following is a more detailed version of the preceding list.

	To File				
From File	**Physical**	**Undefined**	**Printer**	**Diskette**	**Card**
Physical	X	X	X	X	X
Logical	X	X	X	X	X
Diskette	X		X		X
Card	X		X	X	X

Note: An undefined file is a file that has not been created. It is created as a result of a copy operation.

The following shows what copy functions are used for copying part of a file and what type of copy the function is used for (DB=data base file, DEV=device file).

	Type of Copy			
Copy Function	**DB to DB**	**DEV to DB**	**DB to DEV**	**DEV to DEV**
Select records by				
Record number	X	X	X	X
Record key	X		X	
Record character	X	X	X	X
Field content	X		X	
Record format	X			
Number of records	X	X	X	X
Map and drop fields	X			
Insert sequence number and date	X			
Print excluded records	X	X	X	X
Add and replace records	X	X		

Note: For DEV to DB and DB to DEV, sequence numbers and dates are automatically inserted.

The simplest form of copying is copying an entire file. For example, you copy physical file EMP1 to physical file EMP1T. EMP1T is used for testing. Each file contains one member that is named the same as the file.

```
CPYF   FROMFILE(EMP1.PERSONNEL)
       TOFILE(EMP1T.TESTLIB1)
```

You can also copy part of a file to another file. For example, you copy part of a physical file EMP1 to a printer file for the system printer, which is indicated by the predefined option *LIST. You print a listing of all part-time employees.

```
CPYF   FROMFILE(EMP1.PERSONNEL)
       TOFILE(*LIST)
       INCREL(TIME *EQ P)
```

The copy is based on selecting records by the contents of a field. If the field TIME indicates part-time (equals P), the record is copied to the system printer.

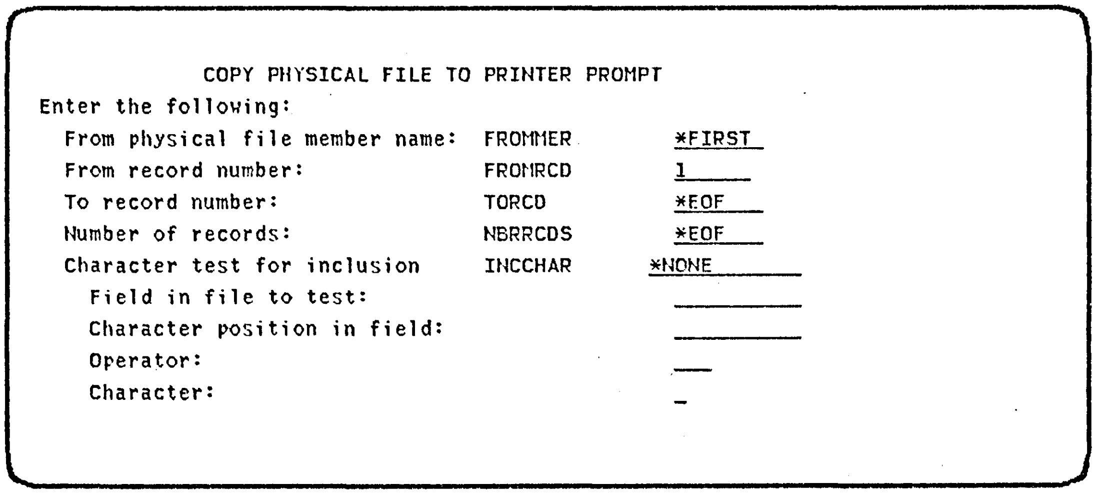
COPY COMMANDS

There are two commands for copying files: Copy File (CPYF) and Copy File Interactive (CPYFI). The CPYF command can be used interactively or in a batch job. It provides the full function for copying files. The CPYFI command provides the same functions as the CPYF command, but it provides a way for you to see only the parameters that are applicable to the type of copy you want. When you enter the CPYFI command, you specify a from file and a to file. System/38 determines what type of fields are involved in the copy and then only prompts you for the parameters that are valid for the type of copy. For example, you enter the following CPYFI command:

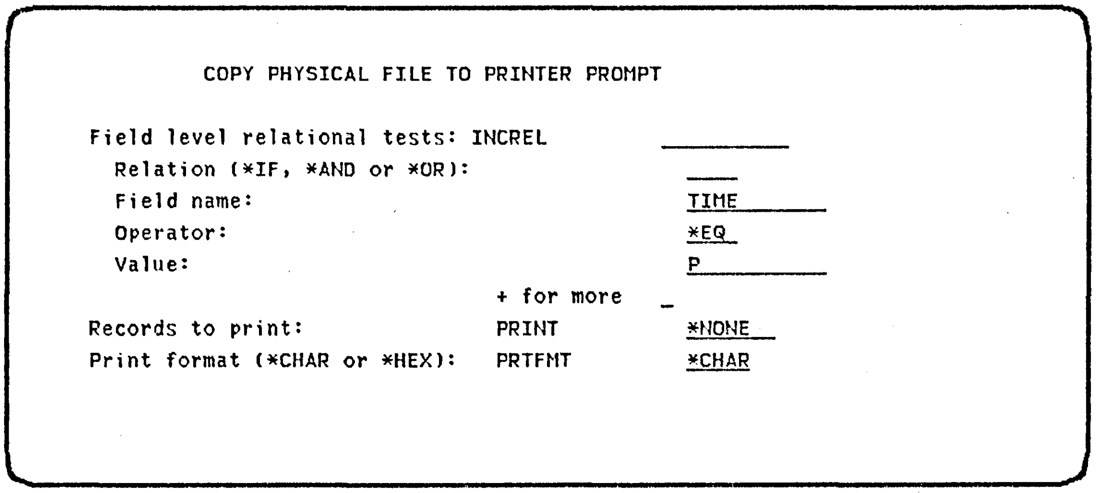
```
CPYFI FROMFILE(EMP1) TOFILE(*LIST)
```

EMP1 is a data base file. You are copying it to the system printer, which is indicated by the predefined option *LIST.

This CPYFI command causes the prompts in Figure 26. You are only prompted for the parameters that apply to a data base to printer type of copy. The INCREL parameter prompt is filled in to show how the previous copy example for copying part of a file to another file is done using interactive copy. The previous example used a CPYF command to print a list of all parttime employees.



```
             COPY PHYSICAL FILE TO PRINTER PROMPT
Enter the following:
  From physical file member name:  FROMMER        *FIRST_
  From record number:              FROMRCD        1_____
  To record number:                TORCD          *EOF___
  Number of records:               NBRRCDS        *EOF___
  Character test for inclusion     INCCHAR       *NONE_______
    Field in file to test:                        __________
    Character position in field:                  __________
    Operator:                                     ___
    Character:                                    _
```



```
             COPY PHYSICAL FILE TO PRINTER PROMPT

    Field level relational tests: INCREL         __________
      Relation (*IF, *AND or *OR):                 ____
      Field name:                                  TIME______
      Operator:                                    *EQ_
      Value:                                       P_________
                                   + for more    _
    Records to print:              PRINT           *NONE__
    Print format (*CHAR or *HEX):  PRTFMT          *CHAR
```

Figure 26. Examples of Copy File Interactive Prompts

COPYING TO AN UNDEFINED FILE

You can copy a data base file to a file that has not been created. The undefined file is created when the copy is performed. The undefined file is created as a physical file and has the same record format as the file from which the copy was made.

You could use this method of copying for the files EMP1 and EMP1T:

```
CPYF   FROMFILE(EMP1) TOFILE(EMP1T)
       CRTFILE(*YES)
```

Member names do not have to be specified because they are always the same as the file names when copying to an undefined file.

MAPPING FIELDS

When copying to a file with a different record format, you can map (copy and convert) the fields to the record format of the file being copied to. For fields that have the same name and attributes, the position of the field in the record format can be changed to the position of the field in the record format it is being copied to. For example, the field CUSNO is in the first position (is the first field) in the record format ORDHDR, but it is in the second position in the record format ORDHD1. When the field is copied, it is mapped into position two of ORDHD1.

If the fields have different attributes but the same names, the fields are mapped according to the following rules.

Originating Field Data Type	**Allowable Length**	**Decimal Positions**	**Mapped To**
Character	1-9999	None	Character
Zoned decimal	1-15 digits	0-9 digits	Zoned decimal, packed decimal, or binary
Packed decimal	1-15 digits	0-9 digits	Zoned decimal, packed decimal, or binary
Binary			
No decimals	1-9 digits	None	Zoned decimal, packed decimal, or binary
With decimals	1-9 digits	1-9 digits	Binary with decimals

When mapping character fields, the field being copied is truncated on the right if it is longer than the field into which the copy is made. For example, a character field of length 10 is copied into a character field of length six; ABCDEFGHIJ becomes ABCDEF. If the field being copied is shorter than the field into which it is copied, the field is padded on the right with blanks. For example, a character field of length 10 is copied into a character field of length 12; ABCDEFGHIJ becomes ABCDEFGHIJƀƀ (ƀ=blank).

When mapping numeric fields and the field being copied is longer than the field into which the copy is made, the field being copied is truncated on the left and right of the decimal point. For example, a zoned decimal field of length nine with four decimal positions is copied to a zoned decimal field of length six with three decimal positions; 00115.1109 becomes 115.110.

When mapping numeric fields and the field being copied is shorter than the field into which the copy is made, the field being copied is padded with zeros on the left and right of the decimal point. For example, a packed decimal field of length seven with five decimal positions is copied to a packed decimal field of length 10 with six decimal positions; 99.99998 becomes 0099.999980.

COMMAND LIST

This is a list of commands related to copying files. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual–CL*.

Descriptive Name	**Command Name**	**Function**
Copy File	CPYF	Copies all or part of a file.
Copy File Interactive	CPYFI	Copies all or part of a file interactively.

Chapter 10. Source Files

A source file is used when a command alone is not sufficient for creating an object. For example, to create a control language program, you must use a source file containing source statements. To create a logical file, you must use a source file containing data description specifications (DDS).

To create the following objects, you can use source files:

- Control language programs (required)
- Files: physical, logical (required), display, and printer
- High-level language programs (required)
- Commands (required)
- Translate tables (required)
- Print images (required)

A source file can be a data base file, card file, diskette file, or inline data file. The file must be defined to contain source and conform to a specific record format.

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

For your convenience, CPF and other program products provide a data base source file for each type of source. These source files are:

File Name	**Used to Create**
QCLSRC	Control language program
QCMDSRC	Command definition statements
QDDSSRC	File
QIMGSRC	Print image
QRPGSRC	RPG program
QTBLSRC	Translate tables
QTXTSRC	Text

You can either add your source members to these files or create your own source files.

Source is processed using a create command to create an object. To create an object you use a create command. For example, to create a physical file you use the Create Physical File (CRTPF) command. A create command specifies through a SRCFILE parameter where the source is stored. Each set of source in a source file is contained in a separate member.

CREATING A SOURCE FILE

To create a source file other than an inline data file you specify the source file type (*SRC) on a create file command:

```
FILETYPE(*SRC)
```

You do not have to describe the record format (through DDS). All source files have the same minimum record format, which consists of three fields:

Field	**Name**	**Data Type and Length**	**Description**
1	SRCSEQ	Zoned decimal, 6 digits, 2 decimal positions	Sequence number for record
2	SRCDAT	Zoned decimal, 6 digits, no decimal positions	Date of last update of record
3	SRCDTA	Character, any length	Data portion of the record

When you create a physical data base source file without using DDS source but by specifying record length (RCDLEN parameter), the source created contains the three fields SRCSEQ, SRCDAT, and SRCDTA. (The record length must include 12 characters for sequence number and update date.) SRCDTA can be defined to contain more than one field (each of which must be character or zoned decimal). If you want to define SRCDTA as containing more than one field, you must define the fields using DDS.

You can use the Source Entry Utility (SEU) Licensed Program 5714-UT1 to enter and update source in a data base file. If you use SEU to enter source in a data base file, SEU adds the sequence number and date to each source record. If you use device files, the system adds them to the source file. In either case, they are generated for you.

When records from a source file are processed, the sequence number and date fields are processed as part of the record. These fields are included in the data received by a program and are included in the data written by a program.

The first sequence number is 0001.00; the next number is 0002.00; and so on. The sequence number is always increased by 1. If you use SEU to update a source file, you can add records between existing records. For example, you add a record between records 0003.00 and 0004.00. The sequence number of the added record is 0003.10.

Source files must be created as keyed sequence files. The sequence number field SRCSEQ is the key field for the source file. Duplicates are sequenced in first-in-first-out (FIFO) order.

When records are intially placed in a source file, the date field is all zoned decimal zeros. If you use SEU, the date field changes in a record as you update the record. If you use device files, the date field is always zeros.

For IBM-supplied data base source files, the data portion length is 80 bytes. For IBM-supplied device source files, the data portion length is the maximum record length for the associated device.

The following Create Physical File (CRTPF) command creates a data base source file:

```
CRTPF FILE(FRSOURCE.QGPL) RCDLEN(92) FILETYPE(*SRC)
```

The record length of 92 includes the 12 characters for the sequence number and date. By default, one member is created for the source file and has the same name as the file.

Source can be placed in the source file by using SEU or by copying from a device file to a physical file. See the *IBM System/38 Source Entry Utility Reference Manual and User's Guide*, SC21-7722, for how to use SEU to enter source.

The following is an example of copying source from a card file to a physical file. The source is on cards in the MFCU. When the source is copied to the file FRSOURCE, sequence numbers and dates are added to the records.

```
CPYF FROMFILE(CARD.QGPL) TOFILE(FRSOURCE.QGPL)
```

CREATING AN OBJECT USING A SOURCE FILE

A create command can create an object using a source file. When you create an object using a source file, you must specify the name of the source file.

The following Create Physical File (CRTPF) command creates the file DSTREF using the data base source file FRSOURCE. The source member is named DSTREF. Because the SRCMBR parameter is not specified, the system assumes that the member name is the same as the file name.

```
CRTPF FILE(DSTREF.QGPL) SRCFILE(FRSOURCE.QGPL)
```

If you specify a device file name as the source file name, source is read from the device (such as cards).

If your create request is spooled, you can use an inline data file as the source file for the request. The inline data file can be either named or unnamed. Named inline data files have a unique file name that is specified on the //DATA command.

Unnamed inline data files are files without unique file names; they are all named QINLINE. The following is an example of an inline data file as a source file.

```
//JOB
CRTPF FILE(ORD199.DSTPRODLB) SRCFILE(QINLINE)
//DATA FILETYPE(*SRC)
   •
   • (source statements)
   •
//ENDJOB
```

In this example, there was no file name specified on the //DATA command. An unnamed spooled file was created when the job was processed by the spooling reader. The CRTPF command must specify QINLINE as the source file name to access the unnamed file. The //DATA command also specifies that the inline file is a source file (*SRC specified for the FILETYPE parameter).

If you specify a file name on the //DATA command, you must specify the same name in the SRCFILE parameter on the CRTPF command. For example,

```
//JOB
CRTPF FILE(ORD199.DSTPRODLB) SRCFILE(ORD199)
//DATA FILE(ORD199) FILETYPE(*SRC)
•
•
•
//ENDJOB
```

If a program requests a spooled file, the file that is read is the first inline file of the specified name. If that file cannot be found, the program reads the first file that is unnamed (QINLINE).

If you do not specify a source file name, an IBM-supplied source file is assumed to contain the needed source data. For example, you are creating a control language program but you did not specify a source file name. The IBM-supplied source file QCLSRC is used. You must have placed the source data in QCLSRC.

If a source file is a data base file, you can specify a source member that contains the needed source data. If you do not specify a source member, the source data must be in a member that has the same name as the object being created.

CHANGING A SOURCE FILE

If you are using SEU to maintain data base source files, see the *SEU Reference Manual and Guide* for how to update data base files. If you are not using SEU to maintain data base source files, you have to delete the old source file and create a new source file.

If your source file is on a diskette, you can copy it to a data base file, change it using SEU, and copy it back to a diskette. If you do not use SEU, you have to delete the old source file and create a new source file.

If you change a source file, the object created from the source file does not match the current source. The old object must be deleted, and a create command must be specified to create the object using the changed source file. For example, if you change the source file FRSOURCE, you have to delete the file DSTREF that was created from the original source file and create it again using the new source file so that DSTREF matches the changed FRSOURCE source file.

Chapter 11. Data Areas

A data area is an object used to pass data between programs within a job and between jobs.

Unlike program variables, data areas are objects and must be created before they can be used in a program or job. A data area can be created as:

- A character string that can be as long as 2000 characters
- A decimal value that can be as long as 15 digits and can contain the digits 0 through 9 with as many as nine decimal positions
- A logical value '0' or '1', where '0' can mean off, false, or no and '1' can mean on, true, or yes

When you create a data area, you can also specify an initial value for the data area. If you do not specify one, the following is assumed:

- 0 for decimal
- Blanks for character
- '0' for logical

To create a data area you use the Create Data Area (CRTDTAARA) command. In the following example you create a data area to pass a customer number from one program to another.

```
CRTDTAARA DTAARA(CUST) TYPE(*DEC)
      LEN(5 0) TEXT('Customer number data area')
```

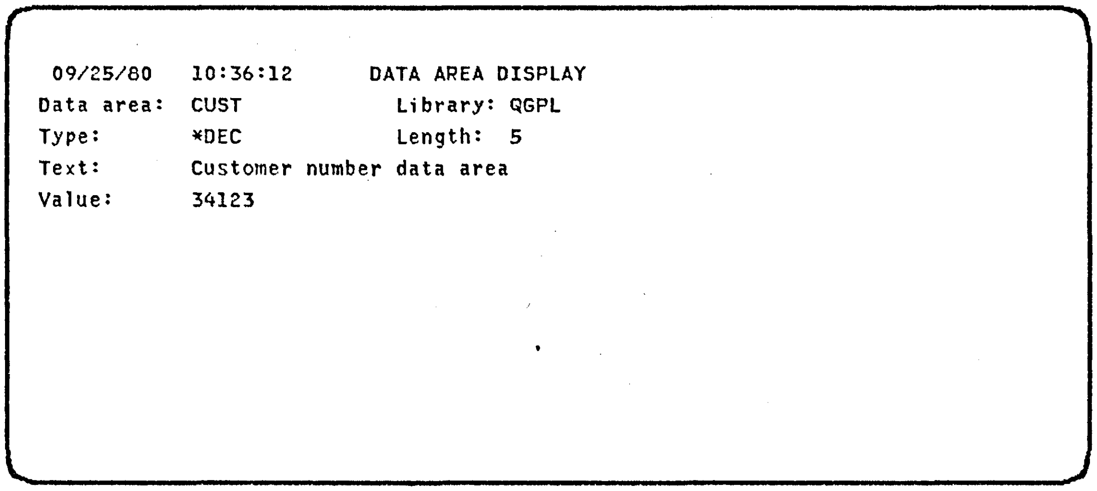
To send or receive a data area in a program, you must declare it to the program with a Declare Data Area (DCLDTAARA) command. When the data area is declared, a control language variable is automatically declared with the same attributes as the data area. The variable name is the data area name preceded by an ampersand (&). For example, the variable declared for the data area CUST is &CUST.

The Send Data Area (SNDDTAARA) and Receive Data Area (RCVDTAARA) commands can be used within your programs to move (send) a variable value to a data area and to move (receive) a data area value to a variable for use by your programs. See Chapter 4, *Control Language Programs* for more information about sending and receiving data areas.

You can display the attributes (name, library, type, length, and text description) of and the value of a data area. The following Display Data Area (DSPDTAARA) command displays the attributes and value of the data area CUST.

```
DSPDTAARA DTAARA(CUST)
```

The resulting display is:



```
 09/25/80    10:36:12      DATA AREA DISPLAY
Data area:   CUST              Library: QGPL
Type:        *DEC              Length:  5
Text:        Customer number data area
Value:       34123
```

The Change Data Area (CHGDTAARA) command changes the current value of a data area; it cannot change the other attributes of the data area.

COMMAND LIST

This is a list of commands related to data areas. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual–CL*.

Descriptive Name	**Command Name**	**Function**
Create Data Area	CRTDTAARA	Creates a data area.
Delete Data Area	DLTDTAARA	Deletes a data area.
Change Data Area	CHGDTAARA	Changes the value of a data area.
Display Data Area	DSPDTAARA	Displays the value and attributes of a data area.

Within Control Language Programs

Descriptive Name	Command Name	Function
Declare Data Area	DCLDTAARA	Declares a data area to a program.
Send Data Area	SNDDTAARA	Moves the contents of a control language variable to a data area.
Receive Data Area	RCVDTAARA	Moves the contents of a data area to a control language variable.

Chapter 12. Message Handling

Messages communicate data between programs, between jobs, between users, and between users and programs. Messages can either be stored independently of or within the programs that use them.

Using System/38 message handling functions, you can:

- Create message files and store messages in them
- Create message queues
- Send messages
- Reply to messages

A message is a communication sent from one system user or program to another. For example, CPF sends a message to the system operator to inform him of some condition that exists in the system, or the system operator sends a message to another work station user to inform him of conditions that affect him. The length, format, content, senders, and receivers of messages vary. Each message is independent of other messages. Unlike records in a data base file, messages in a message file are not related.

There are two types of messages (which are defined in relation to when the messages are created and sent):

- *Predefined messages*. A message that is created and exists outside of the program that uses it. Predefined messages are stored in message files as message descriptions and have unique identifiers for referencing by the program.
- *Impromptu messages*. A message that is created by the sender. An impromptu message is not stored in a message file because it is usually a message that is sent only once. An example of an impromptu message is:

  System going down at 11:00, please sign off

Messages can be defined as:

- Informational (*INFO). A message that conveys information about the condition of a function.
- Inquiry (*INQ). A message that conveys information but also asks for a reply.
- Reply (*RPY). A message that is a response to a received inquiry message.
- Sender's copy (*COPY). A copy of an inquiry message that is kept by the sender.

- Request (*RQS). A message that requests a function from the receiving program. (For example, a control language command can be a request message.)

- Completion (*COMP). A message that conveys completion status of work.

- Diagnostic (*DIAG). A message about errors in the execution of a system function, in an application program, or in input data.

- Escape (*ESCAPE). A message that describes a condition for which a program must terminate abnormally. A program can monitor for the arrival of escape messages from the program it calls or from the machine.

- Notify (*NOTIFY). A message that describes a condition for which a program requires corrective action, or a reply, from its caller. A program can monitor for the arrival of notify messages from the programs it calls.

MESSAGE DESCRIPTIONS

Every predefined message has an associated message description, which is created using the Add Message Description (ADDMSGD) command. The message description can contain:

- Message identifier (required)

- Severity code

- First-level message text (required) with optional substitution variables

- Second-level message text with optional substitution variables

- Description of the format of the message data to be used for the substitution variables

- Validity checking criteria for a reply

- Default value for a reply

- Default message handling action for escape and notify messages

Message Identifier

The message identifier is used to reference the message and is the name of the message description. The message identifier consists of seven characters:

    pppmmnn

ppp is the product or application code and mmnn is the assigned number of the message within the product code. The number specified as mm can be used to further divide a set of product or application messages.

For example,

CPF1234

is message 1234 of CPF.

When you create your own messages, the product code should begin with the letter U. For example,

USR3567

The product code must always be alphabetic; the number must always be numeric.

Severity Code

The severity of a message indicates how important the message is. The severity codes used by CPF for error conditions are:

Code	Description
00	Information only
10	Warning, possible error
30	Input value error, condition ignored or default taken
35	Input value error, requested function terminated
40	Input syntax error, requested function terminated
50	Execution error, requested function terminated
70	Execution error, job terminated
80	System (subsystem, device, or configuration) conditions
99	System termination conditions

First- and Second-Level Messages

A message can consist of two levels of text. The first level is required and identifies the error. The second level is optional and explains the error further or explains the corrective action for the error.

Substitution Variables and Their Associated Message Data Formats

Either level of a message can contain substitution variables. For example,

File &1 not found

contains the substitution variable &1. When the message is sent, the variable &1 is replaced with the name of the file that could not be found. This name is supplied by the sender. For example,

File ORDHDRP not found

Compare this to:

File not found

Substitution variables can make a message more specific.

The substitution variable must begin with & (ampersand) and be followed by any number from 1 to 9.

You must specify the message data format for the substitution variable by specifying data type and length. The valid data types are:

- Quoted character string (*QTDCHAR). A string of character data to be enclosed in apostrophes.
- Unquoted character string (*CHAR). A string of character data not to be enclosed in apostrophes. Trailing blanks are deleted.
- Hexadecimal (*HEX). A string to be preceded by the character X and enclosed in apostrophes; each byte of the string is to be converted into two hexadecimal characters (0 through 9 and A through Z).
- Binary (*BIN). A binary integer (either 2 or 4 bytes long) formatted as a signed decimal integer.
- Decimal (*DEC). A packed decimal number to be formatted as a signed decimal number with a decimal point.
- Time of day (*DTS). An 8-byte system date and time stamp for which the date is to be formatted as specified in the QDATFMT system value and the time is to be formatted as hh:mm:ss.
- System pointer (*SYP). A 16-byte pointer to a system object (see *Glossary*). In a first- or second-level message, the name of the object is formatted the same as the *CHAR type data.
- Space pointer (*SPP). A 16-byte pointer to a program object (see *Glossary*). In a first- or second-level message, the data in the object is formatted the same as the *HEX type data.

Note: The data types *DTS, *SYP, and *SPP are defined to support IBM-supplied messages. You cannot use these types in messages you define.

Validity Checking for Replies

If you create a notify or inquiry message that requires a reply, the reply must be a single value. You can specify validity checking for:

- Type of reply
  - Decimal (*DEC)
  - Character (*CHAR)
  - Alphabetic (*ALPHA)
  - Name (*NAME)

- Maximum length of reply
  - For decimal, 15 (9 decimal positions)
  - For character and alphabetic, 32
  - For name, 10

- Values that can be used for the reply
  - A list of values
  - A list of special values
  - A range of values
  - A simple relation that the reply value must meet

Default Values for Replies

In addition, you can specify a default value for a reply. A default reply must meet the same validity checking criteria as the other replies for the message or be specified as a special value in the message description. A default value is used when there is no one to reply to a message or when a user has indicated that all message replies are to be defaulted (default delivery) instead of being displayed to him immediately.

Default Message Handling for Escape and Notify Messages

For each escape or notify message you can set up a default message handling action for your programs to use if they receive an escape or notify message for which they were not monitoring or for which handling is not specified.

Default message handling actions can consist of:

- Default program name. A program to be called that takes default action to handle a message.

- Dump list. A list of message data field numbers (the same numbers as the substitution variables) that indicate which objects are to be dumped. In addition, you can specify that an entire job structure be dumped.

- Service log indication. Whether the message is to be logged to the service log (see *Message Logging* later in this chapter).

If you do not specify default actions in message descriptions or in your programs, CPF provides standard default actions.

Example of Describing a Message

The content of the message description is specified in the Add Message Description (ADDMSGD) command. In the following example you create a message, to use in your applications such as order entry, that is issued when a customer number entered on the display was not found. The message is:

Customer number &1 not found

The ADDMSGD command for this message is:

```
ADDMSGD  MSGID(USR4310)
         MSGF(USRMSG.QGPL)
         MSG('Customer number &1 not found')
         SECLVL('Change customer number')
         SEV(40)
         FMT((*CHAR 8))
```

MESSAGE FILES

Predefined messages are stored in message files. (Note the MSGF parameter on the preceding ADDMSGD command.) By using predefined messages you can change and translate them into languages other than English without affecting the program that uses them. If messages were within a program, the program would have to be recompiled when the messages were changed.

When you get your system, the CPF messages are stored in the message file QCPFMSG in the library QSYS. In addition, there are message files for each program product you order. All of these message files are in the system library QSYS. You can create message files to contain messages you create. Message file names follow the System/38 naming conventions.

When you create a message file, you can specify the maximum size in K bytes. The following formula can be used to determine the maximum:

$S + (I \cdot N)$

Where	S	is the initial amount of storage
	I	is the amount of storage (increment) to add each time
	N	is the number of times to add storage

The defaults for S, I, and N are 10, 2, and 3, respectively.

For example, you specify S as 5, I as 1, and N as 2. When the file reaches the maximum of 5K, the system automatically adds another 1K bytes to the initial storage. 1K bytes can be added to the storage two times to make the total maximum of 7K bytes.

You use the Create Message File (CRTMSGF) command to create the message file. For example, the following CRTMSGF command creates the message file USRMSG referenced in the preceding ADDMSGD command.

```
CRTMSGF MSGF(USRMSG.QGPL)
        TEXT('Message file for user-created messages')
```

Retrieving Message Text

You can use the Retrieve Message (RTVMSG) command to retrieve for your program the text of a message from a message file. By doing this, you can write the message to an output listing or handle it in other ways. Besides specifying the message identifier and message file name, you can specify:

- Message data fields. The message data for the substitution variables.

- A group of control language variables into which the following information is placed (each corresponds to one variable).
  - First-level message text (character variable)
  - Length of first-level message, including length of substitution variable data (decimal variable)
  - Second-level message text (character variable)
  - Length of second-level message, including length of substitution variable data (decimal variable)
  - Severity code (decimal variable)

The following RTVMSG command copies first-level message text into a CL variable.

```
RTVMSG  MSGID(USR1000) MSGF(USRMSG.QGPL)
        MSGDTA(&FILE &LIB) MSG(&MSG)
```

The message USR1000 is:

```
File &1 not found in library &2
```

The data for &1 is contained in the program variable &FILE and the data for &2 is contained in the program variable &LIB. The message is placed in the CL variable &MSG.

MESSAGE QUEUES

When a message is sent, it is sent to a message queue. The system user or program associated with the message queue receives the message from the queue. Similarly, a reply to a message is sent back to the message queue of the user or program requesting the reply.

The types of message queues are:

- Work station message queues. For sending and receiving messages between work station users and between work station users and the system operator. The name of the queue is the same as the name of the work station. The queue is created when the work station is described to the system.

- System operator message queue (QSYSOPR). For receiving and replying to messages from the system, work station users, and application programs.

- System log message queues. For sending information to the system history log and service log from any job in the system.

- Job message queues. For receiving input to be processed (such as commands) and for sending messages that result from processing the input; the messages are sent to the requester of the job. Job message queues exist for each job and only exist for the life of the job.

- User message queues. For sending messages to system users and between application programs. You must create these queues.

The attributes of a message queue are:

- The maximum amount of storage that can be used for the message queue. The same formula as was used to determine the size of a message file can be used for a message queue (see *Message Files* earlier in this chapter). Size is specified in the Create Message Queue (CRTMSGQ) command.

- How the senders of messages are to be identified. Senders can be identified by any combination of the following or not at all:
  - Job name, user name, and job number. (For interactive jobs, the work station name is used as the job name.)
  - Name of the program sending the message and the instruction number in the program.
  - System date and time stamp.

  The CRTMSGQ command specifies how senders are identified.

- Whether changes to the message queue must be written immediately to the disk unit. This is specified in the CRTMSGQ command.

- The method of delivery for messages arriving at a message queue. When a message queue is created, the method of delivery is defined as hold delivery. The Change Message Queue (CHGMSGQ) command must be used to define delivery as anything other than hold. The types of delivery are:
  - Break delivery. A job is interrupted and the message is delivered.
  - Notify delivery. A work station user is notified by means of the Attention light or audible alarm (or by both) that a message is on the queue. He gets the message by using the Display Message (DSPMSG) command.
  - Hold delivery. The message queue holds the messages until the user asks for them with the Display Message (DSPMSG) command.
  - Default delivery. All messages are ignored, and any messages requiring a reply are sent the default reply for the message.

- How to handle messages for break delivery.
  - Automatically execute the Display Messages (DSPMSG) command. For an interactive job, the messages are displayed at the work station. For a batch job, the messages are listed to a spooled printer file.
  - Invoke a program to handle the messages.

- What the severity code is for filtering messages for break and notify delivery or when the Display Messages (DSPMSG) command is entered. Messages with severity equal to or greater than the minimum severity code specified are displayed. The minimum severity code is specified in the CHGMSGQ command.

Receiving Messages from a Message Queue

A program receives messages through a Receive Message (RCVMSG) command. Messages can be received according to a message reference key or message type or both. A message reference key is assigned to each old message on a message queue. This key is passed as variable data because it is nonprintable. You must declare this variable in your program.

To receive a message you can specify:

- Message queue. Where the message is to be received from.

- Message type. Either a specific message type can be specified or all types can be specified.

- Whether to wait for the arrival of a message. After the wait is over and no message is received, control returns to the requester.

- Whether to remove the message from the message queue after it is received. If it is not removed, it becomes an old message on the message queue and can only be received through its message reference key. (See *Removing Messages from a Message Queue* for more information.)

- A group of control language variables into which the following information is placed (each corresponds to one variable).
  - Message reference key of the message in the message queue
  - First-level message text (character variable)
  - Length of first-level message, including length of substitution variable data (decimal variable of length 5)
  - Second-level message text (character variable)
  - Length of second-level message, including length of substitution variable data (decimal variable of length 5)
  - Message data for the substitution variables provided by the sender of the message (character variable)
  - Length of the message data (decimal variable of length 5)
  - Message identifier (character variable at least seven characters long)
  - Severity code (decimal variable of length 2)
  - Sender of the message (character variable can be as long as 54 characters)
  - Type of reply message received (character variable at least two characters long)
  - Command function key that was pressed in replying to the message (character variable)

The following RCVMSG command specifies that any new message on the program message queue of the requesting program invocation is to be received.

```
RCVMSG PGMQ(*) MSGTYPE(*ANY) MSG(&MSG)
```

The message received is placed in the variable &MSG. * and *ANY are default values for the PGMQ and MSGTYPE parameters, respectively.

Removing Messages from a Message Queue

Messages are held on a message queue until they are removed using a Remove Message (RMVMSG) command, the RMV parameter on the Receive Message (RCVMSG) and Send Reply (SNDRPY) commands, or the remove option of the display messages display. You can remove:

- A single message
- All messages
- All old messages
- All new messages

To remove a single message using the RMVMSG command or the RCVMSG command, you specify the message reference key of the message to be removed. A unique message reference key is assigned to every old message on a message queue. This key is passed as a variable because it is a nonprintable value. You must declare this variable in your program.

Note: The message reference key can also be used to receive a message and to reply to a message.

In the following RMVMSG command you are removing a message from the user message queue JONES. The message reference key is in the control language variable &MRKEY.

```
RMVMSG MSGQ(JONES) MSGKEY(&MRKEY)
```

Sending Messages to a Message Queue

Messages can be sent:

- From one system user to another system user, even if the receiver of the messages is not currently using the system
- From one program to another program
- From a program to a system user, even if the receiver of the messages is not currently using the system

System users can send only impromptu messages and replies. Programs can send impromptu messages, predefined messages, and user-defined data.

Messages Sent by a System User

The command you use to send impromptu messages is Send Message (SNDMSG).

The SNDMSG command sends an information or inquiry message to the system operator message queue (QSYSOPR), a work station message queue, or a user message queue. You can send an information message to more than one message queue at a time. But you can only send an inquiry message to one message queue at a time. The message is delivered according to the delivery type specified for the message queue.

If an inquiry message is sent, you can specify that the reply be sent to a message queue other than that of the work station of the sender of the inquiry messages.

The following SNDMSG command is sent by a work station user to the system operator.

```
SNDMSG  MSG('Mount dkt vol ABCDE1 through ABCDE5')
        TOMSGQ(QSYSOPR)
```

Messages Sent by a Program

The Send Program Message (SNDPGMMSG) command is used by a program to send a message.

The SNDPGMMSG command sends the following types of messages:

- Information
- Inquiry
- Completion
- Diagnostic
- Escape
- Notify

Messages can be sent by a program to the following types of queues:

- External message queue (of the requester of the job)
- Program message queue (of a program invoked by the job)
- System operator message queue
- Work station message queue
- User message queue

To send a message from a program, you can specify the following on the SNDPGMMSG command:

- Message identifier or message text. The message identifier is the name of the message description.

- Message file. The name of the message file containing the message description.

- Message data fields. If a predefined message is sent, these fields contain the values for the substitution variables in the message. The format of each field must be described in the message description. If an impromptu message is sent, there are no message data fields.

- Message queue to receive the message.

- Message type. The following indicates which types of messages can be sent to which types of queues (V = valid).

Message	Message Queue Type				
Type	**External**	**Program**	**QSYSOPR**	**Work Station**	**User**
Information	V	V	V	V	V
Inquiry	V		V	V	V
Completion		V			
Diagnostic		V			
Escape		V			
Notify		V			

- Reply message queue. The name of the message queue to receive the reply to an inquiry message. By default, the reply is sent to the program message queue of the program invocation that sent the inquiry message.

- Whether the message is to be sent to an active (allocated) queue only.

- Key variable name. The name of the control language variable to contain the message reference key for an inquiry message.

To send the message in *Example of Describing a Message* (earlier in this chapter), you would use the following command.

```
SNDPGMMSG   MSGID(USR4310) MSGF(USRMSG.QGPL)
            MSGDTA(&CUSNO) TOPGMQ(*EXT)
            MSGTYPE(*INFO)
```

The substitution variable for the message is the customer number. Because the customer number varies, you cannot specify the exact customer number as a message data field in the MSGDATA parameter. Instead, in your program you declare a program variable for the customer number &CUSNO and then specify this variable as the message data field. When the message is sent the current value of the variable is passed in the message.

In addition, you do not always know which work station is using the program, so you cannot specify the exact work station message queue that the message is to be sent to (TOPGMQ parameter). You specify the external message queue *EXT.

Job Message Queues

Job message queues are created for each job on the system to handle all the message requirements of the job. Job message queues for a single job consist of an external message queue (*EXT) and a set of program message queues. The external message queue is used to communicate with the external requester (such as a work station user) of the job. Messages sent to the external message queue of a job are also written to the job log.

Program message queues are used to send messages between programs of a job. These messages can be request messages, completion messages, diagnostic messages, escape messages, or notify messages. A program message queue is created for each invocation of a program and is given the same name as the program invocation. However, if a program is invoked more than once, only the program message queue of the most current invocation can be used for message handling. That is, when program A sends a message to program B, the message goes to the program message queue for the last invocation of program B.

A message can be sent to the caller of a program without specifying either the caller or the program name. The caller is always the previous program invocation in the invocation stack. In addition, a message can be sent to the caller of a specified program. In which case, the program name must be specified.

A program message queue for a program is deleted when an invocation is deleted and is no longer available for use.

Figure 27 shows the relationship of program invocations, the job message queue, and the program message queues. The dashed line (———) indicates which message queue is associated with which invocation of a program.

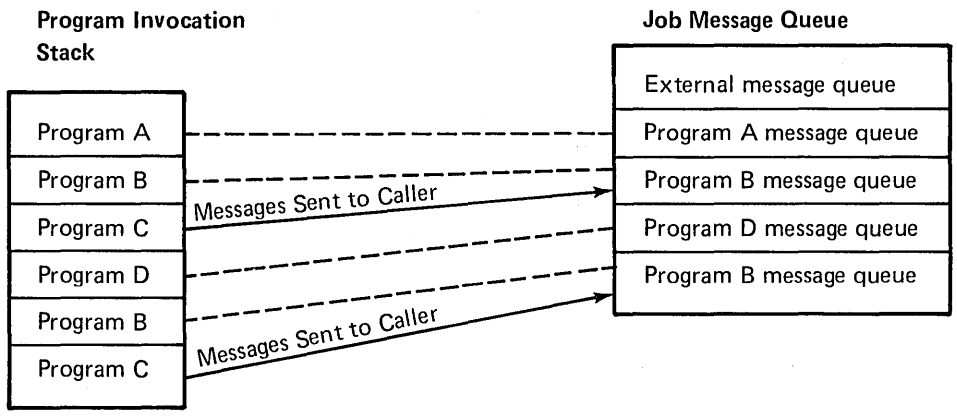


Figure 27. Relationship of Program Invocations and the Job Message Queue

In Figure 27 program B has two program message queues, one for each invocation of the program. There are no message queues for program C because no messages were sent to C.

The following are the types of messages that can be sent to a program message queue.

- Information messages
- Request messages
- Completion messages
- Diagnostic messages
- Escape and notify messages

Request Messages

Receiving request messages is a simple method for a control language program to obtain input from a work station and to display resulting diagnostics and completion messages. A program receives a request from its program message queue and sends messages that result from the request to the program message queue. The program displays these messages and prompts the work station user from the next request.

Request messages can only be received interactively. A work station user is prompted to enter a request for the execution of some function by your program. The request is placed on the program's message queue.

The syntax of the data in the request must be defined by your program. Your program must interpret the request and diagnose errors. For example, the control language commands are requests that are received by the CPF interpretive control language processor for interpretation and diagnostics.

Completion and Diagnostic Messages

A control language program can send diagnostic and completion messages to its caller's program message queue. These messages tell the caller of errors detected by the program and tell the caller the status of work being done by the program. Normally, an escape message is sent to the caller's program message queue to tell the caller that diagnostic messages were sent. For a completion message, an escape message is not sent because the requested function was performed.

Escape and Notify Messages

A control language program can send escape and notify messages to its caller's program message queue. An escape message tells the caller that the program terminated abnormally and why. The caller does not return control to the program. See *Monitoring and Handling Escape Messages* for more information about using escape messages.

A notify message tells the caller that the program will terminate processing unless a reply to the condition is returned to the program. If the caller returns control to the program, the program can receive the reply. See *Monitoring and Handling Notify Messages* later in this chapter.

Monitoring and Handling Escape Messages

You can monitor for escape messages that are sent to your control language program's program message queue. Escape messages can be sent by the commands in your program or by the programs your program calls. Escape messages are sent by these programs to tell your program of an error condition that forced the sender to terminate. By monitoring for escape messages, you can take corrective actions or clean up and terminate your program.

The *Message Guide* contains information about each IBM-defined escape message. You should keep a list of all messages that you have defined.

Use the Monitor Message (MONMSG) command to monitor for escape messages. To monitor escape messages you must specify message identifiers for the messages in one of the following ways:

- pppmmnn

  Monitors for a specific message. For example, MCH1211 is the message identifier of the zero divide escape message.

- pppmm00

  Monitors for any message with an identifier that begins with a specific program product (ppp) and the digits specified by mm.

- ppp0000

  Monitors for every message with an identifier that begins with a specific program product (ppp). For example, CPF0000 indicates that all escape messages beginning with CPF are monitored.

- Special value

  Monitors for all escape messages based on a special value. For example, *ZRODVD indicates that the escape message for the zero divide condition (MCH1211) is to be monitored for.

In addition to monitoring for escape messages by message identifier, you can compare a character string, which you specify in the MONMSG command, to data sent in the message. For example, the following command monitors for an end-of-file escape message for the file MYFILE. The name of the file is sent as message data.

```
MONMSG MSGID(*ENDF) CMPDTA(MYFILE) EXEC(GOTO EOJ)
```

The compare data can be as long as 28 characters, and the comparison starts with the first character of the first field of the message data. If the compare data matches the message data, the action specified in the EXEC parameter is executed. The message data fields are documented in the *Message Guide* for each message.

You can monitor for an escape message sent by a *specific* command in your program by specifying the MONMSG command immediately following the command. You can use as many as five MONMSG commands for a single command. This lets you handle different escape messages in different ways.

The EXEC parameter specifies how an escape message is to be handled. Any command except PGM, ENDPGM, IF, ELSE, DCL, ENDDO, and MONMSG can be specified in the EXEC parameter. You can specify a DO command in the EXEC parameter, in which case, the commands in the do-group are executed. When the command or do-group (in the EXEC parameter) has been executed, control returns to the command in your program that is after the command that sent the escape message. However, if you specify a GOTO or RETURN command, control does not return. If you do not specify the EXEC parameter, the escape message is ignored and your program continues.

You can also monitor for an escape message sent by any command in your program by specifying the MONMSG command immediately following the last declare command in your program. This lets you handle the same escape message in the same way for all commands. The EXEC parameter must be specified. Only the GOTO command can be specified in the EXEC parameter.

You can also monitor simultaneously for the same escape message to be sent by a specific command in your program *and* by any command. This requires two MONMSG commands. One MONMSG command follows the command that needs special handling for the escape message; for that command, this MONMSG command is used when the escape message is sent. The other MONMSG command follows the last declare command so that for all other commands, this other MONMSG command is used.

The following shows an example of a CHGVAR (Change Variable) command being monitored for a zero divide escape message, message identifier MCH1211.

```
CHGVAR VAR(&A) VALUE(&A/&B)
MONMSG MSGID(MCH1211) EXEC(CHGVAR VAR(&A) VALUE(1))
```

The value of the variable &A is changed to the value of &A divided by &B. If &B equals 0, the divide operation cannot be done and the zero divide escape message is sent to the program. When this happens you change the value of &A to 1 (as specified in the EXEC parameter).

Many escape messages can be sent to your program by commands and programs it calls. Probably, you will not want to monitor for and handle all of these. Normally, only a few escape messages pertain to the function of your program; the rest of those that can be sent should never actually be sent. CPF provides default monitoring and handling of any messages you do not monitor.

Default handling assumes that an error has been detected in your program. If you are debugging the program, the message is sent to your work station. Then you can enter commands to analyze and correct the error. If you are not debugging the program, a dump is taken and the function check escape message (CPF9999) is sent to your program. You can monitor for function check escape messages so that you can either:

1. Clean up and terminate the program

2. Continue with some other aspect of your program

If you do not monitor for a function check escape message, your program is terminated by CPF and the function check message is sent to the program's caller.

Monitoring and Handling Notify Messages

Besides monitoring for escape messages, you can monitor for notify messages that are sent to your control language program's program message queue by the commands in your program or by the programs it calls. Notify messages are sent by these programs to tell your program of a condition that is not typically an error. By monitoring for notify messages, you can specify an action different from what you would specify if the condition had not been detected.

Monitoring and handling notify messages is similar to monitoring and handling escape messages. The difference is in what happens if you do not monitor and handle messages. Unlike escape messages, unmonitored notify messages are not considered an indication of an error in your program. Instead, for notify messages, the default reply stored in the message description is used to tell the sender of the message what to do.

MESSAGE LOGGING

There are two types of logs for messages:

- Job log
- System log

A job log contains information related to requests entered for a job. System logs contain system data such as security and service information.

Job Log

Each job has an associated job log. The job log can contain:

- The commands in the job (not commands in control language programs)
- All messages sent to the requester
- All replies to the messages sent to the requester
- All completion and diagnostic messages sent to and not removed from the program message queue

At the end of the job, the job log is written to an output file so that it can be printed. After the job log is written to the output file, the job log is deleted.

You can control what is written in the job log by specifying the message logging level, message logging severity, and message text logging level in the job description. (See *Job Descriptions* in Chapter 16, *Work Management.*) There are five message logging levels:

Level	Description
0	No data is logged
1	All messages sent to the external message queue including indications of when the job started and ended and the status of completion
2	Logging level 1, any message with a severity greater than or equal to the specified severity, and any request for which a diagnostic message was issued that has a severity greater than or equal to the specified severity
3	Logging level 1, all requests, and any message with a severity greater than or equal to the specified severity
4	All requests and all messages

You can log only the first-level message or both the first- and second-level messages.

System Logs

There are three system logs:

- History log
- Service log
- Programming change log

The history log contains system, subsystem, and job information, device status, and system operator messages. The service log contains information about errors detected in IBM program products. The program change log contains information about the application of program changes to IBM products.

The sizes of the history and service logs are specified in the system values QSRVLOGSIZ and QHSTLOGSIZ, respectively. When a log is full, that version of the log can be saved. (See Chapter 19, *Save/Restore* for information on how to save an object.) A new version of the log is automatically created. Each version is a physical file that is named in the following manner:

Qxxxyydddn

Where	xxx	is a three-character description of the log type (HST for history; SRV for service; CHG for programming change)
	yyddd	is the Julian date on which the log version was created
	n	is a sequence number within the Julian date (0 through 9 or A through Z)

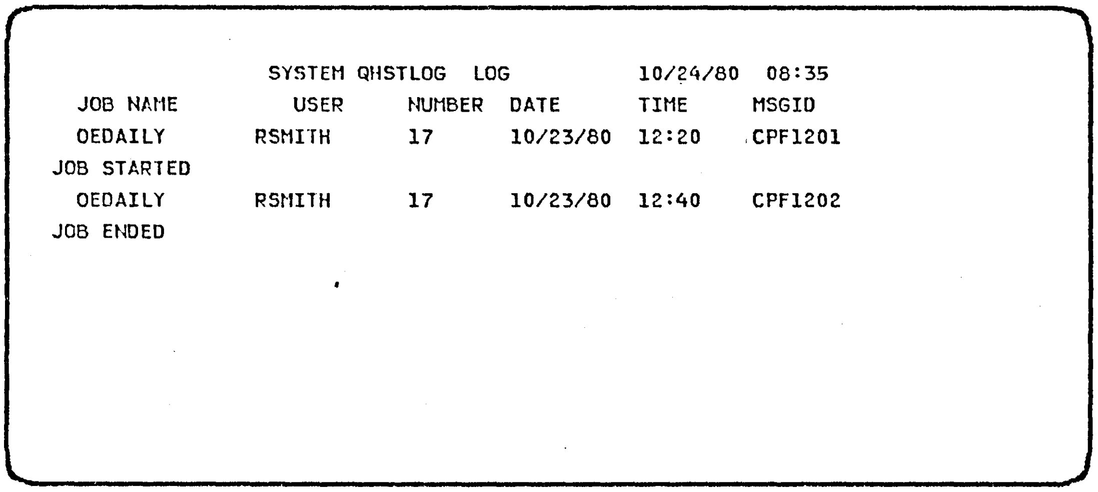
You can display or print the information in a log using the Display Log (DSPLOG) command. You can select the information you want displayed or printed by specifying any combination of the following:

- Log (QHSTLOG, QSRVLOG, or QCHGLOG)
- Period of time
- Name of job for which entries are to be displayed
- Message identifiers of entries that are to be displayed

The following DSPLOG command displays all the available entries for the job OEDAILY in the current day.

```
DSPLOG JOB(OEDAILY)
```

The resulting display is:



```
                    SYSTEM QHSTLOG  LOG            10/24/80  08:35
  JOB NAME            USER      NUMBER  DATE       TIME      MSGID
  OEDAILY           RSMITH      17      10/23/80   12:20     CPF1201
JOB STARTED
  OEDAILY           RSMITH      17      10/23/80   12:40     CPF1202
JOB ENDED
```

COMMAND LIST

This is a list of commands related to message handling. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL.*

Messages

Descriptive Name	Command Name	Function
Send Message	SNDMSG	Sends an impromptu message to a message queue.
Remove Message	RMVMSG	Removes a message from a message queue.
Display Message	DSPMSG	Displays messages from a message queue.

Message Queues

Descriptive Name	Command Name	Function
Create Message Queue	CRTMSGQ	Creates a message queue.
Delete Message Queue	DLTMSGQ	Deletes a message queue.
Change Message Queue	CHGMSGQ	Changes the attributes of a message queue.

Message Files

Descriptive Name	Command Name	Function
Create Message File	CRTMSGF	Creates a message file.
Delete Message File	DLTMSGF	Deletes a message file.
Add Message Description	ADDMSGD	Adds a message description to a message file.
Remove Message Description	RMVMSGD	Removes a message description from a message file.

Commands Within Programs

Descriptive Name	Command Name	Function
Send Program Message	SNDPGMMSG	Sends a message from within a program to a message queue.
Receive Message	RCVMSG	Receives a message from a message queue.
Send Reply	SNDRPY	Sends a reply for a message to a message queue.
Retrieve Message	RTVMSG	Retrieves a message from a message file.
Monitor Message	MONMSG	Monitors for escape and notify messages sent to a program's program message queue.

Chapter 13. Defining Commands

You can define your own commands to call your applications. By defining your own commands, you can set up default values for parameters, perform validity checking outside of your application programs, and define prompt text for using commands interactively. You can also create your own versions of IBM-supplied commands.

The alternative to defining commands is to use the CALL command to call your applications. However, validity checking is not performed on the parameters passed in the CALL command when the call is entered. A program would have to do the validity checking. If you define your own commands, CPF performs the validity checking.

All commands entered using spooling or the Source Entry Utility (Licensed Program 5714-UT1) are checked syntactically using the command definition. You can do additional parameter checking by including the checking in the command processing program (CPP) you write for the command (see *Writing a Command Processing Program* later in this chapter) or by writing a program called a validity checker. If you use a validity checker, the command is checked before it is passed to the CPP for processing. If an error is found, a message is issued to the user so errors can be corrected immediately. The CPP can assume that the data passed to it is correct.

Validity checking using the command definition can ensure that:

- The required parameters are entered.
- Each parameter value meets data type and length requirements.
- Each parameter value meets the requirements of:
  - A list of valid values
  - A range of values
  - A relational comparison to a value
- Conflicting parameters are not entered.

HOW TO DEFINE COMMANDS

To create a command you must first define the command through command definition statements. Command definition statements let you:

- Define the keyword and parameter values for a parameter (PARM statement)
- Define prompt text for the command prompt (CMD statement) and for the parameters (PARM statement)

The statements are entered into a source file so they can be used with the Create Command (CRTCMD) command to create a command. See Chapter 10, *Source Files* for entering source statements.

Defining Parameters

You can define as many as 50 parameters for each command. To define a parameter you must use the Parameter (PARM) statement. You name the keyword for the parameter and the type of parameter value that can be passed. The keyword can be up to 10 alphameric characters, the first of which must be alphabetic. The basic parameter types are (TYPE parameter value given in parentheses):

- Decimal (*DEC). The parameter value is a decimal number.
- Logical (*LGL). The parameter value is a logical value, '1' or '0'.
- Character (*CHAR). The parameter value is a character string, which can be enclosed in apostrophes.
- Name (*NAME). The parameter value is a character string of which the first character is alphabetic and the remaining characters are alphameric.
- Generic name (*GENERIC). The parameter value is a generic name. A generic name consists of a set of characters that identify a group of objects and ends with an * (asterisk). For example, INV* identifies the objects INV, INVOICE, and INVENTORY.
- Variable name (*VARNAME). The parameter value is a variable name passed as a character string. A variable is a name that references an actual data value during execution. A variable name can be as long as 10 alphameric characters (the first of which must be alphabetic) preceded by an & (ampersand)–for example, &PARM.
- Date (*DATE). The parameter value is a character string that is passed to the CPP in the format gyymmdd (g = guard digit, y = year, m = month, d = day). CPF sets the guard digit based on the year in the specified date. The guard digit is 0 if yy equals a number from 70 through 99; it is 1 if yy equals a number from 00 through 69. The system values QDATFMT and QDATSEP determine what format the date must be specified in on the command.
- Time (*TIME). The parameter value is a character string that is passed to the CPP in the format hhmmss (h = hour, m = minute, s = second).

You can also specify a length for any parameter value except date or time. (Date is always 7 characters and time is always 8 characters.) There are restrictions as to the maximum length that can be specified for a value and there are default lengths if you do not want to specify a length. The following shows the maximum and default lengths for each parameter type that you can specify a length for.

Data Type	Default Length	Maximum Length
*DEC	15 5 decimal positions	15 9 decimal positions
*LGL	1	1
*CHAR	32	2000
*NAME	10	256
*GENERIC	10	256
*VARNAME	11	11

If you are defining an optional parameter, you can define a value to be used if the parameter value is not specified on the command. This value is called a *default value*. The default value must meet all the value requirements for that parameter (such as type, length, and special values). If you do not specify a default value for an optional parameter, the following default values are used:

Data Type	Default Value
*DEC	0
*LGL	'0'
*CHAR	Blanks
*NAME	Blanks
*GENERIC	Blanks
*VARNAME	Blanks
*DATE	Zeros
*TIME	Zeros

The following is information you must consider when defining parameters. The associated PARM statement parameter is given in parentheses.

- Whether a value is returned by the CPP (RTNVAL)
- Whether the parameter is not to appear externally to the user but is to be passed to the CPP as a constant (CONSTANT)
- Whether the parameter is restricted (RSTD) to specific valid values or can include any value that matches the parameter type, length, value range, and a specified relationship
- What the specific valid parameter values are (VALUES, SPCVAL, and SNGVAL)
- What tests should be performed on the parameter value to determine its validity (REL and RANGE)
- Whether the parameter is optional or required (MIN)
- How many values can be specified for a parameter that requires a list (MIN and MAX)
- Whether the value is a file name (FILE)

- Whether the value must be the exact length specified (FULL)

- Whether the length of the value should be returned with the value (VARY)

- Whether attribute information should be returned about the value passed for the parameter (PASSATR)

- What the prompt text for the parameter is (PROMPT)

In the following example you define a parameter OETYPE for a command to call your order entry application.

```
PARM  KWD(OETYPE)  TYPE(*CHAR)  RSTD(*YES)
      VALUES(DAILY WEEKLY MONTHLY)  MIN(1)
      PROMPT('Type of order entry:')
```

The OETYPE parameter is required (MIN parameter is 1) and its value must be DAILY, WEEKLY, or MONTHLY.

Defining the Prompt for the Command Name

When a user chooses to be prompted for a command instead of entering the command, the command name (such as ORDENTRY) and the word PROMPT are automatically displayed on line 1. You can specify an additional character string, called prompt text, to appear on line 1 before the command name. The prompt text must be a character string of, at the most, 30 characters.

The Command (CMD) statement is used to define the prompt. For example, you enter the following for the user-defined ORDENTRY.

```
CMD PROMPT('ORDER ENTRY')
```

Line 1 of the prompt looks like this:

```
ORDER ENTRY (ORDENTRY) PROMPT
```

Creating Commands

You use the Create Command (CRTCMD) command to create a command. When you create a command, you can define the following attributes of the command.

- The validity checker

- What mode the command can execute in:
  - Production
  - Debug
  - Service

- Where the command can be used:
  - Batch job stream
  - Interactive job
  - Control language program in a batch job stream
  - Control language program in an interactive job
  - As a command interpretively executed by the system through the QCAEXEC interface

- What file the prompt text is kept on

For example, you define a command named ORDENTRY to call your order entry application. The Create Command (CRTCMD) command defines the preceding attributes for ORDENTRY and creates the command using the parameter definitions contained in the IBM-supplied source file QCMDSRC.

```
CRTCMD  CMD(ORDENTRY.DSTPRODLB)
        PGM(ORDENT.DSTPRODLB)
        PUBAUT(*NONE)
        TEXT('Calls order entry application')
```

The resulting command is:

```
ORDENTRY  OETYPE(value)
```

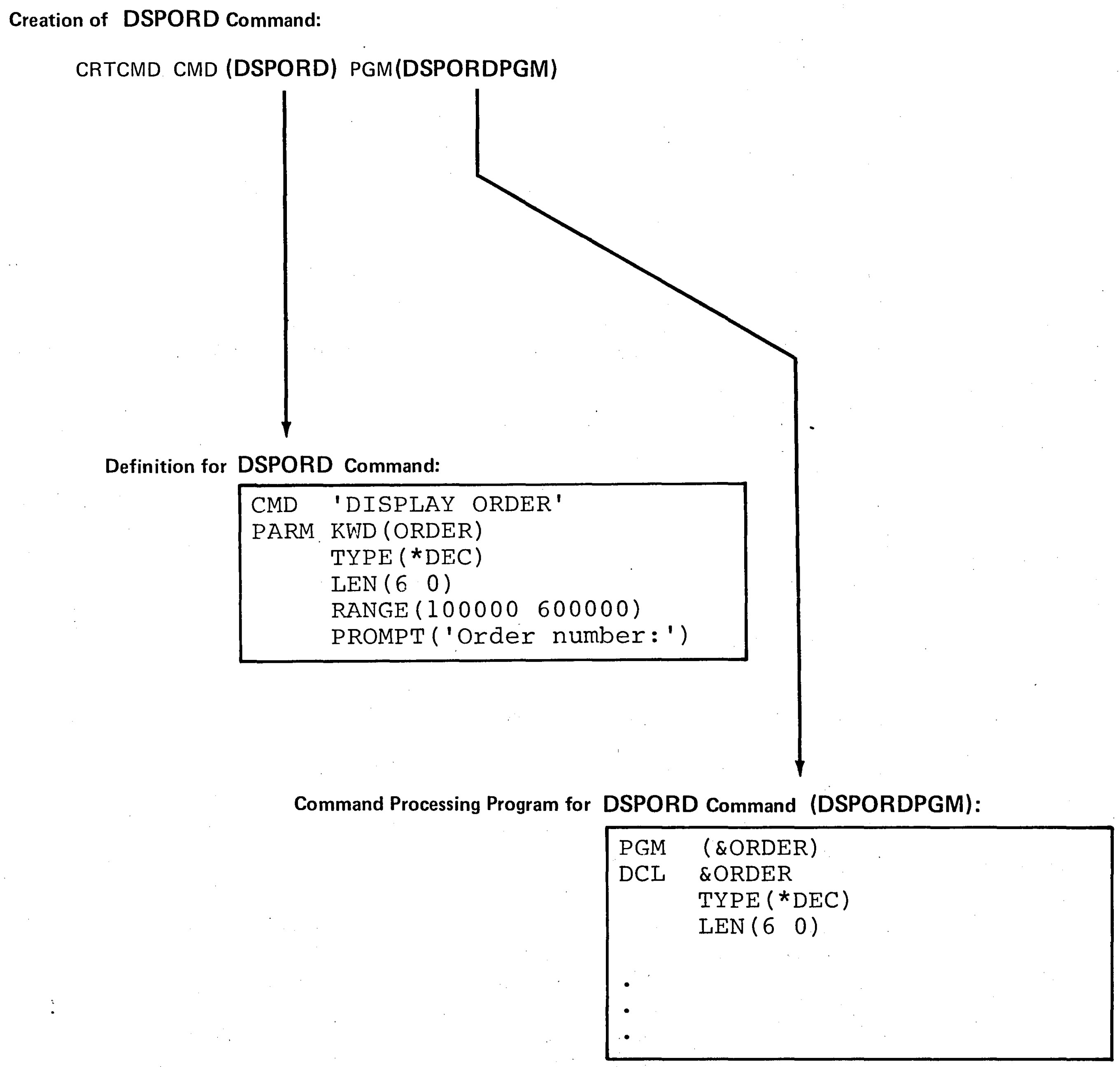
where the value can be DAILY, WEEKLY, or MONTHLY.

WRITING A COMMAND PROCESSING PROGRAM

Writing a command processing program (CPP) is much like writing any HLL or control language program. Messages issued as a result of executing the CPP can be sent to the job message queue and automatically displayed. You can send displays to the requesting display work station.

Notes:
1. The parameters defined in the command are passed in a parameter list in the order they were defined (the PARM statement order).
2. Decimal values are passed as packed decimal values of the length specified in the PARM statement.
3. Character, name, and logical values are passed as a character string of the length defined in the PARM statement.

Figure 28 shows the relationship between the Create Command (CRTCMD) command, the command definition statements, and the CPP.



Creation of DSPORD Command:

```
CRTCMD CMD(DSPORD) PGM(DSPORDPGM)
```

Definition for DSPORD Command:

```
CMD   'DISPLAY ORDER'
PARM  KWD(ORDER)
      TYPE(*DEC)
      LEN(6 0)
      RANGE(100000 600000)
      PROMPT('Order number:')
```

Command Processing Program for DSPORD Command (DSPORDPGM):

```
PGM   (&ORDER)
DCL   &ORDER
      TYPE(*DEC)
      LEN(6 0)

.
.
.
```

Figure 28. Command Relationships

If the CPP is a control language program, the variables that receive the parameter values must be declared to correspond to the type and length specified for each PARM statement. The following shows this correspondence. (Note the declare for the parameter ORDER in Figure 28.)

PARM Statement		**DCL (Declare) Command**	
Type	**Length**	**Type**	**Length**
*DEC	x y	*DEC	x y
*LGL	1	*LGL	1
*CHAR	n	*CHAR	$\leq n$[1]
*NAME	n	*CHAR	$\leq n$[1]
*GENERIC	n	*CHAR	$\leq n$[1]
*DATE	n	*CHAR	$\leq n$[1]
*TIME	n	*CHAR	$\leq n$[1]

[1]Must equal n if RTNVAL(*YES) is specified.

REDEFINING IBM-SUPPLIED COMMANDS

You can redefine IBM-supplied commands to conform to a language other than English or to terminology that suits your installation better. You can change the command name, the keyword name, some of the parameter values, and the prompt text. Even though you can change some parts of the commands, you cannot change the attributes of the parameters passed to the CPP. No matter what part of a command you are changing, you must use command definition statements to redefine it and the CRTCMD command to create the new command. If the command name does not change, the command must be placed in a different library from the IBM-supplied command. The CPP used for the IBM-supplied command is the same one used for the new command.

Note: If you change the prompt text for the command, you must compile again programs that contain the command.

You can change the name of a command without changing anything else in the command. If you do so, you specify a different command name on a CRTCMD command and use the same source member as was used for the IBM-supplied command. The source member name for an IBM-supplied command is the same as the command name.

```
CRTCMD  CMD(MOVE)  PGM(QLIMVOBJ)  SRCFILE(QCMDSRC)
        SRCMBR(MOVOBJ)  TEXT('Move object')
```

The IBM-supplied MOVOBJ (Move Object) command was changed to MOVE. The source member MOVOBJ is in the IBM-supplied source file QCMDSRC.

Besides changing command names you can change keyword names. To change a keyword name, you specify a new keyword name on a PARM statement and change all references to the keyword name on other command definition statements.

Similarly, if you want to change a parameter value to a constant value so that you never need to enter the parameter, you specify the constant value on the CONSTANT parameter of the PARM statement.

You can specify that if a certain value is entered for a parameter value that another value actually be sent to the CPP. Primarily, this is useful for translating IBM-supplied commands into a language other than English. You can translate a command without rewriting the CPP to handle non-English words. The English version can be passed to the CPP. The SPCVAL parameter is used to define what values can be specified and what values are actually passed to the CPP.

You can also restrict parameter values using the VALUES parameter on the PARM statement and change default values using the DFT parameter on the PARM statement.

Commands to Which New Special Values Cannot Be Added

New special values (SPCVAL parameter) cannot be added to the following commands. (These commands are handled specifically by the control language compiler.)

CALL	IF
CHGVAR	MONMSG
CNLRCV	PGM
DCL	RCVDTAARA
DCLDTAARA	RCVF
DCLF	RETURN
DO	SNDDTAARA
ELSE	SNDF
ENDDO	SNDRCVF
ENDPGM	TFRCTL
GOTO	WAIT

COMMAND LIST AND STATEMENT SUMMARY

This is a list of commands related to defining commands. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL.*

Descriptive Name	Command Name	Function
Create Command	CRTCMD	Creates a command from a command definition.
Delete Command	DLTCMD	Deletes a command.
Change Command	CHGCMD	Changes the attributes of a command definition.

This is a list of command definition statements related to defining commands.

Descriptive Name	Statement Name	Function
Command	CMD	Specifies the prompt text for a command name.
Dependent	DEP	Defines the relationship between parameters.
Element	ELEM	Defines the elements in a list used as a parameter value.
Parameter	PARM	Defines a parameter for a command.
Qualifier	QUAL	Defines a qualified name used as a parameter value.

Chapter 14. Application Documentation

Documentation aids provided on System/38 are useful when designing and maintaining application programs.

The following commands provide information on the description, organization, and usage of data base and device files.

- Display Data Base Relations (DSPDBR)
- Display File Description (DSPFD)
- Display File Field Description (DSPFFD)
- Display Program References (DSPPGMREF)

The information provided by these commands can be used to determine what effect changes in an application might have on files and programs:

- What DDS and program source needs to be changed
- What files have to be created again
- What files are used by a given program

The output from the DSPDBR, DSPFFD, and DSPPGMREF commands can be placed in a physical file member for which you can build your own access path for analysis of the output. The physical file is created for you when you request that output be placed in the file. (The request is made using the OUTFILE parameter on the DSPDBR, DSPFFD, and DSPPGMREF commands.) The record format for the file is one supplied by CPF (Figure 29). Initially, the file is created with private authority. Only the owner can use it. However, you can authorize other users to use the file (see *Granting Authority* in Chapter 18, *Security*).

Once the physical file has been created, you can use it again for the same command. However, the file is cleared before new output is entered. For example, you specify a DSPDBR command with the output going to the file OUTANYLS. Later, you specify another DSPDBR command with the output going to OUTANLYS. OUTANLYS is cleared before the output is entered; the output from the first DSPDBR command is gone.

Data Base Relations:

```
     A*
     A* DATA BASE RELATIONS OUTPUT RECORD QWHDRDBR
     A*
     A          R QWHDRDBR                  TEXT('DATA BASE RELATIONS')
     A            WHBRFI        10A         COLHDG('FILE')
     A            WHBRLI        10A         COLHDG('LIBRARY')
     A            WHBRMB        10A         COLHDG('MEMBER')
     A            WHBRRD        10A         COLHDG('RECORD FORMAT')
     A            WHBNO          4B 0       COLHDG('DEPENDENT COUNT')
     A            WHBDTM        13A         COLHDG('DATE/TIME')
     A            WHBREFI       10A         COLHDG('DEPENDENT FILE')
     A            WHBRELI       10A         COLHDG('LIBRARY')
     A            WHBREMB       10A         COLHDG('MEMBER')
     A            WHBTYPE        1A         COLHDG('TYPE OF DEPENDENCY')
     A
     A
     A
     A
     A
```

Program References:

```
     A*
     A* DATA BASE RECORD FORMAT QWHDRPPR
     A*
     A          R QWDDRPPR                  TEXT('PROGRAM REFERENCES')
     A            WHPLIB        10A         COLHDG('LIBRARY')
     A            WHPPNAM       10A         COLHDG('PROGRAM')
     A            WHPTEXT       50A         COLHDG('TEXT DESCRIPTION')
     A            WHPFNUM        4B 0       COLHDG('OBJECT COUNT')
     A            WHPDTTM       13A         COLHDG('DATE TIME')
     A            WHPFNAM       11A         COLHDG('OBJECT')
     A            WHPLNAM       11A         COLHDG('LIBRARY')
     A            WHPSNAM       10A         COLHDG('SOURCE FILE NAME')
     A            WHPRFNO        4B 0       COLHDG('RECORD FORMAT COUNT')
     A            WHPFUSG        4B 0       COLHDG('FILE USAGE')
     A            WHPRFNM       10A         COLHDG('RECORD FORMAT')
     A            WHPRFSN       13A         COLHDG('SEQUENCE')
     A            WHPRFFN        8B 0       COLHDG('FIELD COUNT')
     A
     A
     A
```

Figure 29 (Part 1 of 2). Record Formats for Output File

File Field Descriptions:

```
     A*
     A* FORMAT/FIELD DATA BASE RECORD QWHDRFFD
     A*
     A          R QWHDRFFD                  TEXT('FORMAT/FIELD DB RECORD')
     A            WHFILE        10          COLHDG('FILE')
     A            WHLIB         10          COLHDG('LIBRARY')
     A            WHCRTD         7          COLHDG('FILE CREATION DATE')
     A            WHCNT          4B 0       COLHDG('RECORD FORMAT COUNT')
     A            WHDTTM        13          COLHDG('DATE/TIME')
     A            WHNAME        10          COLHDG('RECORD FORMAT')
     A            WHSEQ         13          COLHDG('SEQUENCE')
     A            WHTEXT        50          COLHDG('TEXT DESCRIPTION')
     A            WHFLD#         4B 0       COLHDG('FIELD COUNT')
     A            WHRLEN         4B 0       COLHDG('RECORD LENGTH')
     A            WHFLDI        10          COLHDG('INTERNAL FIELD NAME')
     A            WHFLDE        10          COLHDG('EXTERNAL FIELD NAME')
     A            WHFOBO         4B 0       COLHDG('OUTPUT BUFFER OFFSET')
     A            WHIBO          4B 0       COLHDG('INPUT BUFFER OFFSET')
     A            WHFLDB         4B 0       COLHDG('FIELD LENGTH')
     A            WHFLDD         4B 0       COLHDG('DECIMAL DIGITS')
```

```
     A            WHFLDP         4B 0       COLHDG('DECIMAL PRECISION')
     A            WHFTXT        50          COLHDG('TEXT DESCRIPTION')
     A            WHRCDE         4B 0       COLHDG('CHANGE CODE') TEXT('1-OTHER-
     A                                      , 2-VALIDITY, 4-EDIT, 8-DECIMALS, 1-
     A                                      6-LENGTH, 32-TYPE, 64-NAME, 128-LEN-
     A                                      GTH')
     A            WHRFIL        10          COLHDG('REFERENCE FILE')
     A            WHRLIB        10          COLHDG('REFERENCE LIBRARY')
     A            WHRFMT        10          COLHDG('REFERENCE RECORD FORMAT')
     A            WHRFLD        10          COLHDG('REFERENCE FIELD')
     A            WHCHD1        20          COLHDG('COLUMN HEADING 1')
     A            WHCHD2        20          COLHDG('COLUMN HEADING 2')
     A            WHCHD3        20          COLHDG('COLUMN HEADING 3')
     A            WHFLDT         1          COLHDG('FIELD TYPE') TEXT('B-BINARY-
     A                                      , A-CHARACTER, S-ZONED, P-PACKED')
     A            WHFIOB         1          COLHDG('I/O ATTRIBUTE') TEXT('I-INP-
     A                                      UT, O-OUTPUT, B-BOTH')
     A            WHECDE         1          COLHDG('EDIT CODE')
     A            WHVC#E         4B 0       COLHDG('VALIDITY CHECK COUNT')
     A
```

Figure 29 (Part 2 of 2). Record Formats for Output File

DISPLAYING DATA BASE RELATIONS

You can display or print the following information about the organization of your data base.

- A list of all data base files (physical and logical) that use a record format
- A list of all data base files (physical and logical) that depend on the specified file for data sharing or access path sharing
- A list of all members (physical and logical) that depend on the specified file for data sharing or access path sharing

In the following example, you display a list of all authorized data base files associated with physical file ORDHDRP, and use the record format ORDHDR. You need this information because you are going to make changes in the record format and you want to ensure that the files affected can use the changed record format. You use the Display Data Base Relations (DSPDBR) command to display the list.

```
DSPDBR FILE(ORDHDRP.DSTPRODLB) RCDFMT(ORDHDR)
```

The resulting display is:

```
  10/23/80        DATA BASE RELATIONS
Command input:
  Generic file name:             FILE      ORDHDRP
  Library name:                            DSTPRODLB
  Member name:                   MBR       *NONE
  Generic record format name:    RCDFMT    ORDHDR
  Output file name:              OUTFILE   *NONE
  Library name:                            *NONE
  Output:                        OUTPUT    *
File name:          ORDHDRP      Library:  DSTPRODLB
  Member name:                             ORDHDRP
```

```
  Record format name:                            ORDHDR
  Number of dependencies:                        3
FORMAT DEPENDENT:
  Dependent file name:                       Library:
     ORDFILL                                 DSTPRODLB
     ORDHDRL                                 DSTPRODLB
     ORDHDRL1                                DSTPRODLB
```

DISPLAYING FILE DESCRIPTIONS

You can display or print file descriptions for a single file in a specified library or all files of the same name in all libraries. The types of displays for a file are

- File attributes
- Access path specifications (logical and physical files only)
- Select/omit specifications (logical files only)
- Alternate collating sequence specifications (physical and logical files only)
- Record format specifications
- Member attributes (logical and physical files only)
- Spooling attributes (device files only)

In the following example you display the file attributes for the physical file ORDHDRP. You use the Display File Description (DSPFD) command.

```
DSPFD FILE(ORDHDRP.DSTPRODLB) TYPE(*ATR)
```

The resulting display is:

```
DATA BASE FILE ATTRIBUTES:
  Access path type:                      ARRIVAL
  Share the file:              SHARE     *NO
  Maintenance:                 MAINT     *IMMED
  Alternate collating sequence:ALTSEQ-DDS NO
  Duplicate rule:              UNIQUE-DDS NO
  Key order:                   LIFO-DDS   FIFO
```

```
Recover access path:          RECOVER     *NO
Maximum wait time for file:   WAITFILE    *IMMED
Force ratio:                  FRCRATIO    *NONE
Deleted records threshold:    DLTPCT      *NONE
Check level of format:        LVLCHK      *YES
Maximum number of members:    MAXMBRS     1
Member count:                             1
Maximum key length:
Key position count:
Allocate space:               ALLOCATE    *NO
Contiguous space:             CONTIG      *NO
```

```
Unit identifier:              UNIT        *ANY
Size of members:              SIZE
  Initial number of records:              10000
  Increment value:                         1000
  Maximum times to increment:                 3
Record capacity:                          13000
```

DISPLAYING FILE FIELD DESCRIPTIONS

You can display field information for both data base and device files:

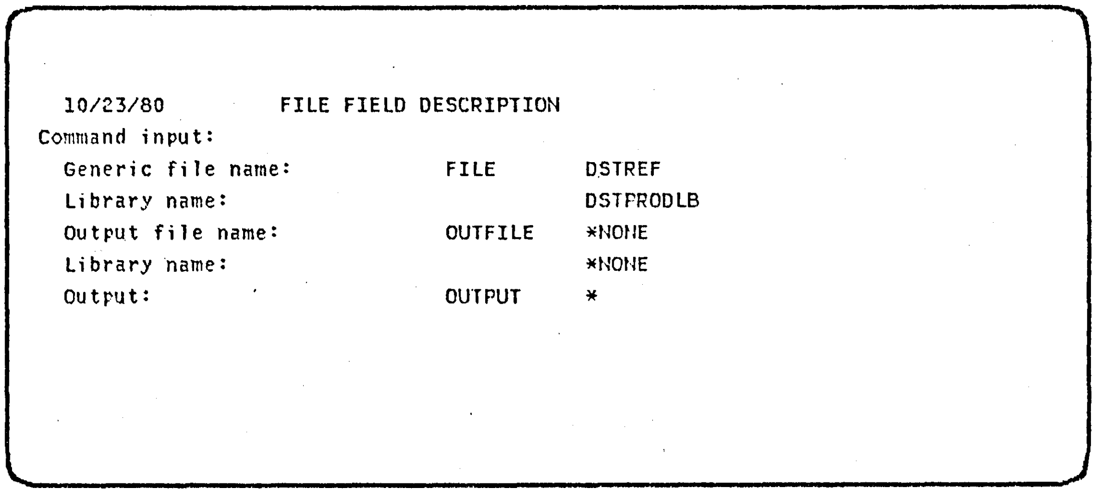
- A single file
- A group of files
- All files in a single library
- All files in all libraries

This field information can be placed in a data base file for which you can build your own access path for the analysis of data.

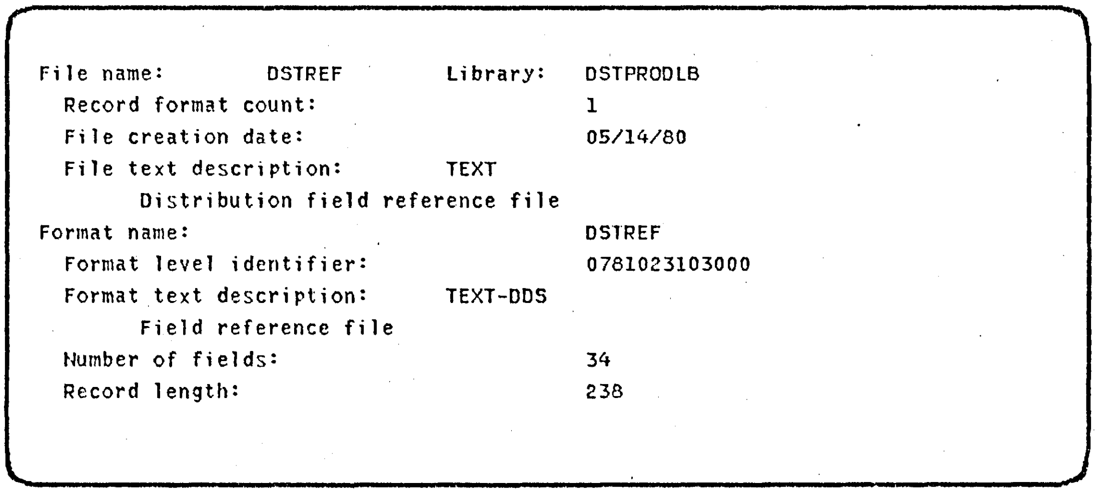
In the following example you display the field information for your field reference file DSTREF. You use the Display File Field Description (DSPFFD) command.

```
DSPFFD FILE(DSTREF.DSTPRODLB)
```

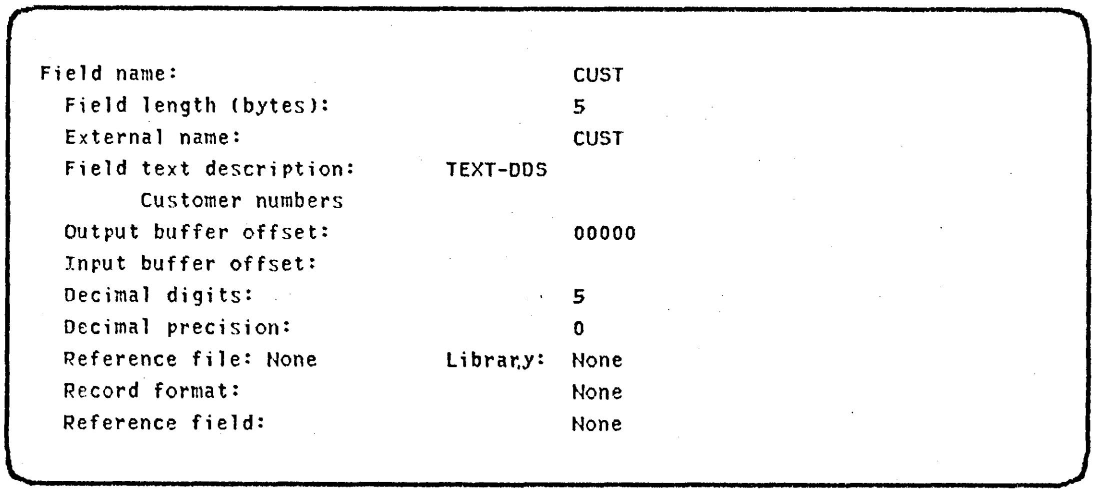
The resulting display for one of the fields is:



```
 10/23/80        FILE FIELD DESCRIPTION
Command input:
 Generic file name:            FILE       DSTREF
 Library name:                            DSTPRODLB
 Output file name:             OUTFILE    *NONE
 Library name:                            *NONE
 Output:                       OUTPUT     *
```



```
File name:        DSTREF       Library:   DSTPRODLB
 Record format count:                     1
 File creation date:                      05/14/80
 File text description:        TEXT
       Distribution field reference file
Format name:                              DSTREF
 Format level identifier:                 0781023103000
 Format text description:      TEXT-DDS
       Field reference file
 Number of fields:                        34
 Record length:                           238
```



```
Field name:                               CUST
 Field length (bytes):                    5
 External name:                           CUST
 Field text description:       TEXT-DDS
       Customer numbers
 Output buffer offset:                    00000
 Input buffer offset:
 Decimal digits:                          5
 Decimal precision:                       0
 Reference file: None          Library:   None
 Record format:                           None
 Reference field:                         None
```

```
Attributes changed:        None
Field type:                S
I/O attribute:
Column heading 1:          CUSTOMER
Column heading 2:          NUMBER
Column heading 3:
```

DISPLAYING PROGRAM REFERENCES

You can display or print file usage and other object (data areas and programs) usage information for:

- A single program
- A group of programs
- All programs in a library
- All programs in all libraries

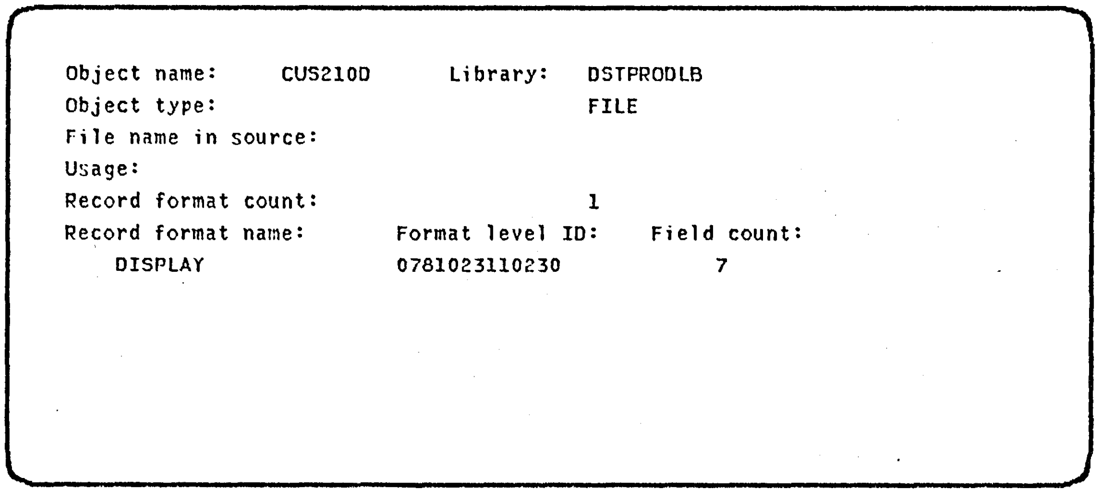
This information can be placed in a data base file for which you can build your own access for analysis of data.

In the following example you display the file usage of the program CUS210.

```
DSPPGMREF PGM(CUS210.DSTPRODLB)
```

The resulting display is:

```
 10/23/80        PROGRAM REFERENCES
Command input:
 Program name:                PGM        CUS210
 Library name:                           DSTPRODLB
 Output file name:            OUTFILE    *NONE
 Library name:                           *NONE
 Output:                      OUTPUT     *
Program name:       CUS210    Library:   DSTPRODLB
 Program text description:
       Customer inquiry
 Program reference count:                1
```



```
Object name:     CUS210D      Library:   DSTPRODLB
Object type:                             FILE
File name in source:
Usage:
Record format count:                     1
Record format name:      Format level ID:    Field count:
    DISPLAY              0781023110230           7
```

This information is displayed for each file used by CUS210.

OTHER DOCUMENTATION AIDS

The following are additional documentation aids:

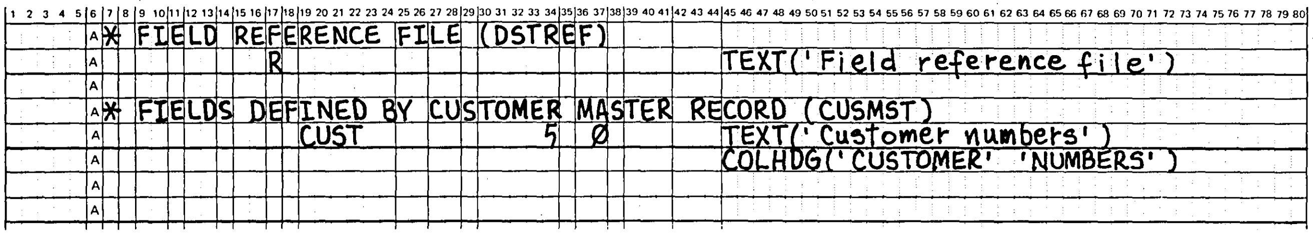
- The TEXT and COLHDG keywords for description of record formats and fields in DDS.
- The * (asterisk) for comments in DDS.
- The TEXT parameter on the create commands.
- The /* */ for comment statements in your programs.

The TEXT keyword can be used to describe a record format or a field within the DDS for a file. The text description cannot exceed 50 characters in length and must be enclosed in apostrophes. This text is used by compilers and by utilities such as the Query Utility (part of the Interactive Data Base Utilities, licensed program 5714-UT1).

The COLHDG keyword is used to specify a column heading on a display for a field. You can specify three lines of column headings for a field; each line can contain 20 characters. Each heading line must be enclosed within apostrophes. Column headings are used by the Query Utility.

An * in position 7 of the DDS form indicates that the line is a comment. Comments can appear anywhere in the file description.

The following shows the use of an *, the TEXT keyword, and the COLHDG keyword. The TEXT keywords describe a record format and a field.



```
     A* FIELD REFERENCE FILE (DSTREF)
     A          R                           TEXT('Field reference file')
     A
     A* FIELDS DEFINED BY CUSTOMER MASTER RECORD (CUSMST)
     A            CUST           5  Ø       TEXT('Customer numbers')
     A                                      COLHDG('CUSTOMER' 'NUMBERS')
     A
     A
```

The TEXT parameters on the create commands are also used to document your application. For example, when you create an object you specify a text description:

```
CRTPF  FILE(DSTREF.DSTPRODLB) SRCFILE(FRSOURCE.QGPL)
       TEXT('Distribution field reference file')
```

This text description becomes part of the object description. You can display this object description to find out the attributes of the object (in this case, the distribution field reference file).

When you want to write comments in your control language programs or append comments to commands in your programs, use the symbols /* */. The comment is written between these symbols and must not exceed 50 characters. For example, in the program created in Chapter 4, *Control Language Programs*, comments were written on commands:

```
        PGM /*ORD040C Order dept general menu*/
        DCLF FILE(ORD040CD)
START:  SNDRCVF RCDFMT(MENU).
        IF(&RESP=1) THEN(CALL CUS210)
        /*Customer inquiry*/
        ELSE
          IF(&RESP=2) THEN(CALL ITM210)
          /*Item inquiry*/
          ELSE
            IF(&RESP=3) THEN(CALL CUS210)
            /*Customer name search*/
            ELSE
              IF(&RESP=4) THEN(CALL ORD215)
              /*Orders by cust*/
              ELSE
                IF(&RESP=5) THEN(CALL ORD220)
                /*Existing order*/
                ELSE
                  IF(&RESP=6) THEN(CALL ORD410C)
                  /*Order entry*/
                  ELSE
                    IF(&RESP=7) THEN(RETURN)
        GOTO START
        ENDPGM
```

COMMAND LIST

This is a list of commands related to documentation aids. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL*.

Descriptive Name	Command Name	Function
Display Program References	DSPPGMREF	Displays a list of files and objects used by a specified compiled program.
Display Data Base Relations	DSPDBR	Displays data base cross reference information: • What data base files use a specified record format. • What data base files depend on a specified file. • What data base members depend on a specified member.
Display File Description	DSPFD	Displays a description of a file: • File attributes • Access path specifications • Select/omit specifications • Alternate collating sequence specifications • Record format specifications • Member attributes • Spooling attributes
Display File Field Description	DSPFFD	Displays field descriptions for a specified file.

Chapter 15. Testing

Testing lets you debug your programs in an environment identical to your normal processing environment without modifying files in production libraries and without modifying a program specifically for testing.

You can use objects in production libraries while you are in debug mode. You do not have to create and store in a test library a special copy of an object such as a program, message queue, or data area to use it in debug mode. The program you are testing can be in a production library.

No special commands specifically for testing are contained within the program being tested. The same program being tested can be run normally without modification. All test commands are specified within the job the program is in, not as a permanent part of the program being tested. In addition, the test functions are only applicable to the job they are set up in. The same program can be used concurrently in another job without being affected by the testing functions set up.

There are three major testing functions you can use on your programs:

- Setting breakpoints to halt program execution
- Specifying traces to trace HLL statement execution
- Displaying and modifying the values of program variables

A *breakpoint* is a place in a program where the system halts execution of the program and gives control to the user (at a work station or through a batch program). At a breakpoint the user can enter commands to request other functions such as displaying and changing program variables. A *trace* is the process of recording the sequence in which the statements in a program are executed. In addition, a record can be kept of the values of the program variables used in the statements.

ENTERING DEBUG MODE

To begin testing you must use the ENTDBG command. This command places you in debug mode and also specifies:

- Which programs are to be debugged
- Which program serves as the default program for the debug job

- Whether a record of the breakpoints is to be written to the job log
- How many statements can be traced for a job and what action the system should take when the maximum number is reached
- Whether production files can be updated during debug

For a program to be debugged it must either be specified on the ENTDBG command or added to the debug job with an Add Program (ADDPGM) command. You can specify as many as 10 programs to be debugged simultaneously in a job.

If you specified 10 programs for debug (using either the ENTDBG or ADDPGM command or both commands) and you want to add more programs to the debug job, you will have to remove some of the previously specified programs. Use the Remove Program (RMVPGM) command. All breakpoints and traces defined for a program being removed are also removed.

When you enter debug mode you can specify that a program be a default program. By specifying a default program you can use breakpoint and trace commands without having to specify a program name each time a command is used. This is helpful if you are only debugging one program. For example, in the Add Breakpoint (ADDBKP) command you would not specify a program name for the PGM parameter because the default program is assumed to be the program the breakpoint is being added to. The default program name must be specified in the list of programs to be debugged (PGM parameter). If more than one program is listed to be debugged, you can specify the default program in the DFTPGM parameter. If you do not, the first program in the list in the PGM parameter is assumed to be the default program.

The default program can be changed by using either the Change Debug (CHGDBG) or the Add Program (ADDPGM) command.

If you log a record of the breakpoints in the job log, the information logged is a condensed form of the breakpoint displays. The job log can be used for offline debugging. The logging level in the job description for the job must be level 3, which means that all information is kept. If the level is other than 3, this information is not kept. You can change the logging level using the Change Job (CHGJOB) command.

You can specify a maximum number of statement executions that can be traced for a job. You can change this maximum using the Change Debug (CHGDBG) command. As long as the maximum has not been reached, you can lower the maximum; otherwise, you can only raise the maximum. If you do not specify a maximum, only 200 statement executions are traced.

When the maximum is reached, the system performs one of the following actions (depending upon what you specify).

- Stops the trace (*STOPTRC).
  - For an interactive job, control is given to you (a breakpoint occurs), and you can remove some of the trace definitions (RMVTRC command) or clear the trace data (CLRTRCDTA command).
  - For a batch job, the trace definitions are removed and the program continues to execute.

- Continues the trace but only the last maximum number of traces can be viewed in the trace output (*WRAP).

By protecting production files from being updated, you can avoid unintentional modification of the files. To protect files you must specify *YES in the UPDPROD parameter.

The following ENTDBG command places the high-level language program CUS310 in debug mode. (CUS310 is a customer master file maintenance program.) CUS310 is also the default program. Data base files in production libraries cannot be updated. The default of 200 statements that can be traced is taken, and the system is to stop tracing but continue executing.

```
ENTDBG PGM(CUS310.DSTPRODLB)
```

Note: You can end debug mode at any breakpoint by entering the End Debug (ENDDBG) command.

ADDING BREAKPOINTS TO PROGRAMS

After you have entered debug mode you can add breakpoints to the program you want debugged. Adding a breakpoint to a program consists of specifying either the statement label or the statement number. When you add a breakpoint to a program, you can also specify program variables whose values you want to display when the breakpoint is reached.

When a breakpoint is about to execute, program execution stops. For an interactive job, the system displays what breakpoint the program has stopped at and, if requested, the values of the program variables. After you get this information (in a display) you can enter commands to request other functions such as changing a variable (CHGPGMVAR command), adding a breakpoint (ADDBKP command), or adding a trace (ADDTRC command). You can enter any control language command that can be used interactively. If you entered a command, you must enter the Resume Breakpoint (RSMBKP) command to resume program execution. If you did not enter a command, you can use command function key 10 to resume execution.

For a batch job, a breakpoint program can be invoked when a breakpoint is reached. You must create this breakpoint program to handle the breakpoint information. The breakpoint information is passed to the breakpoint program. The breakpoint program can invoke another user-created program such as a control language program that can contain the same commands (requests for function) that you would have entered interactively for an interactive job. Any function valid in a batch job can be requested. When the breakpoint program completes executing, the program being debugged continues executing.

To add a breakpoint to a program, use the Add Breakpoint (ADDBKP) command. You can specify 10 statement numbers in one ADDBKP command. The program variables specified on an ADDBKP command apply only to the breakpoints specified on the same command. Only 10 variables can be specified in one ADDBKP command.

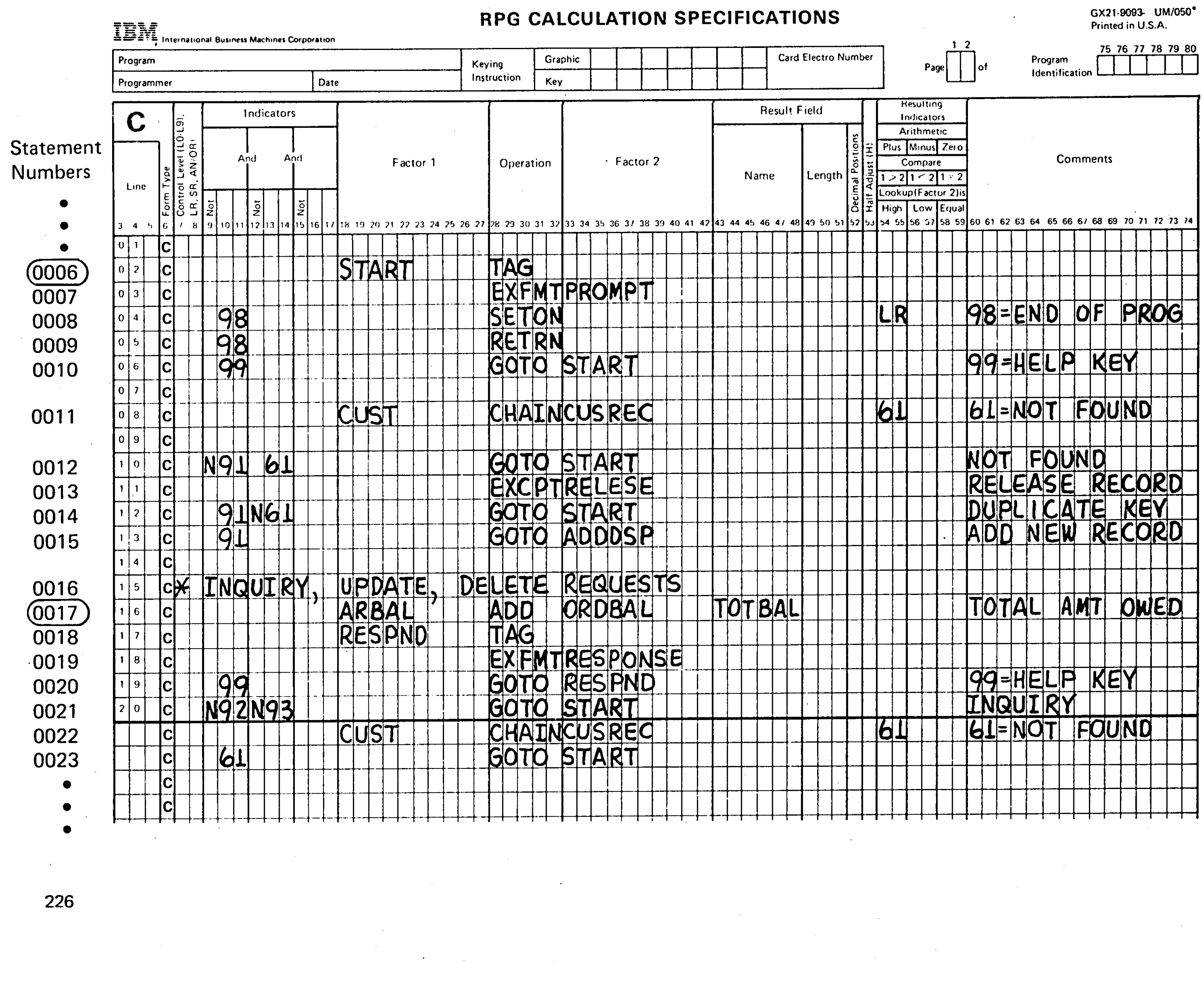
In the ADDBKP command you can also specify the breakpoint program name for a breakpoint reached in a batch job, and the name of the program to which the breakpoint is added. If you do not specify the name of the program to which the breakpoint is added, the breakpoint is added to the default program specified in the ENTDBG or CHGDBG command.

The following ADDBKP commands add breakpoints to the program CUS310. The value of the variable ARBAL is to be displayed when the second breakpoint is reached.

```
ADDBKP   STMT(6)
ADDBKP   STMT(17) PGMVAR(ARBAL)
```

CUS310 is the default program, so it did not have to be specified.

The specifications for CUS310 look like this:



IBM International Business Machines Corporation

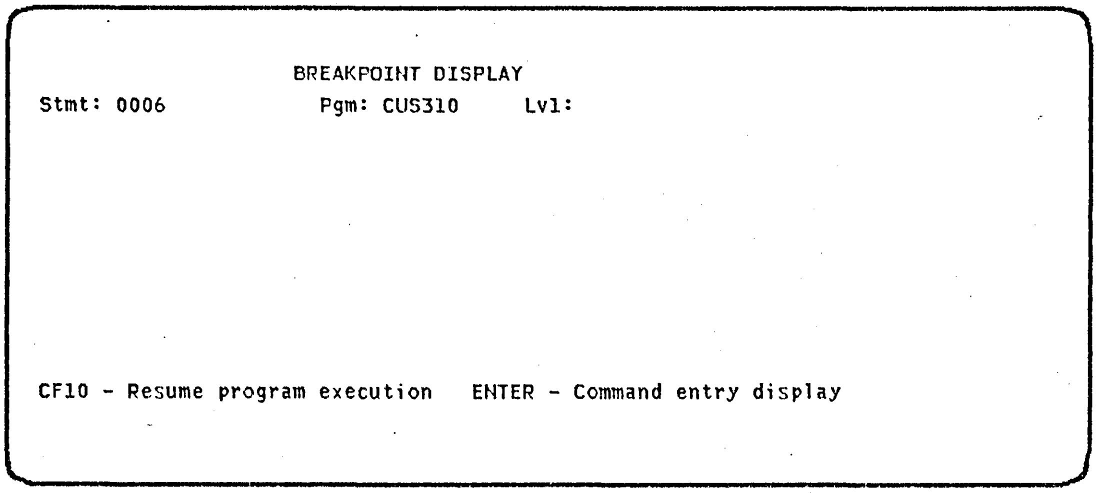
RPG CALCULATION SPECIFICATIONS

GX21-9093- UM/050* Printed in U.S.A.

Program	Keying Instruction	Graphic	Card Electro Number	Page of
Programmer	Date	Key		Program Identification

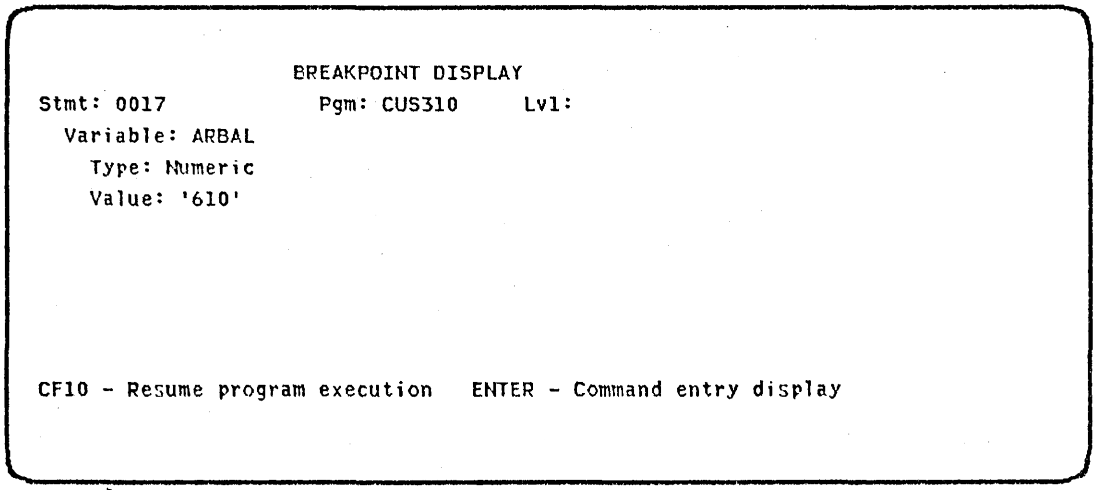
Statement Numbers	Line	Form Type	Indicators	And	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Resulting Indicators High	Low	Equal	Comments
•	01	C												
0006	02	C				START	TAG							
0007	03	C					EXFMT	PROMPT						
0008	04	C	98				SETON				LR			98=END OF PROG
0009	05	C	98				RETRN							
0010	06	C	99				GOTO	START						99=HELP KEY
	07	C												
0011	08	C				CUST	CHAIN	CUSREC			61			61=NOT FOUND
	09	C												
0012	10	C	N91	61			GOTO	START						NOT FOUND
0013	11	C					EXCPT	RELESE						RELEASE RECORD
0014	12	C	91	N61			GOTO	START						DUPLICATE KEY
0015	13	C	91				GOTO	ADDDSP						ADD NEW RECORD
	14	C												
0016	15	C*	INQUIRY,			UPDATE,	DELETE	REQUESTS						
0017	16	C				ARBAL	ADD	ORDBAL	TOTBAL					TOTAL AMT OWED
0018	17	C				RESPND	TAG							
0019	18	C					EXFMT	RESPONSE						
0020	19	C	99				GOTO	RESPND						99=HELP KEY
0021	20	C	N92	N93			GOTO	START						INQUIRY
0022		C				CUST	CHAIN	CUSREC			61			61=NOT FOUND
0023		C	61				GOTO	START						
•		C												
•		C												
•														

The first breakpoint just tells where you are in the program. The following is displayed as a result of reaching the first breakpoint.



```
                     BREAKPOINT DISPLAY
Stmt: 0006             Pgm: CUS310     Lvl:









CF10 - Resume program execution   ENTER - Command entry display
```

The following is displayed as a result of reaching the second breakpoint.



```
                     BREAKPOINT DISPLAY
Stmt: 0017             Pgm: CUS310     Lvl:
  Variable: ARBAL
    Type: Numeric
    Value: '610'






CF10 - Resume program execution   ENTER - Command entry display
```

At this point you could change the value of one of these variables to alter your program's execution. You use the Change Program Variable (CHGPGMVAR) command to change the value of a variable.

Returning to a Breakpoint in One Program from Another Program

When you are executing a program in debug mode, you may find yourself in the following situation.

- You reached a breakpoint in one program.
- At that breakpoint you called another program that was also in debug mode.
- You reached a breakpoint in the second program.
- Now you want to return to the first program.

To return to the breakpoint you specify the name of the first program. For example, a breakpoint occurred in the control language program ORD045C. At the breakpoint you called another program CUS310. At a breakpoint in CUS310 you want to go back to the breakpoint in ORD045C. The following command gets you back to the breakpoint in ORD045C.

```
RTNBKP PGM(ORD045C)
```

The invocation of CUS310 disappears when this RTNBKP command is executed. At the breakpoint you returned to, you can request more functions or resume program execution (RSMBKP command).

ADDING TRACES TO PROGRAMS

A trace differs from a breakpoint in that you are not given control during the trace. The system records the traced statements executed. However, the trace information is not automatically displayed when the program completes execution. You must request a display of the trace information. The display shows the sequence in which the statements were executed and, if requested, the values of the variables used in the statements.

Adding a trace consists of specifying what statements are to be traced and, if you want, the names of program variables used in the statements. When a traced statement executes and a specified variable is in that statement, the value of the variable is recorded. Also, you can specify that the values of the variables are to be recorded only if they have changed from the last time a traced statement was executed.

To specify which statements are to be traced, you can specify

- The statement number at which the trace is to start and the statement number at which the trace is to stop
- That all statements in the program are to be traced
- A single statement number of a statement to be traced

However, you can only specify a total of five statement ranges for a single program, which is a total taken from all the Add Trace (ADDTRC) commands for the program. In addition, only ten variables can be specified for each statement range.

The following Add Trace (ADDTRC) command adds a trace to the program CUS310. The value of the variable TOTBAL is to be recorded only if its value changes between the times each traced statement is executed.

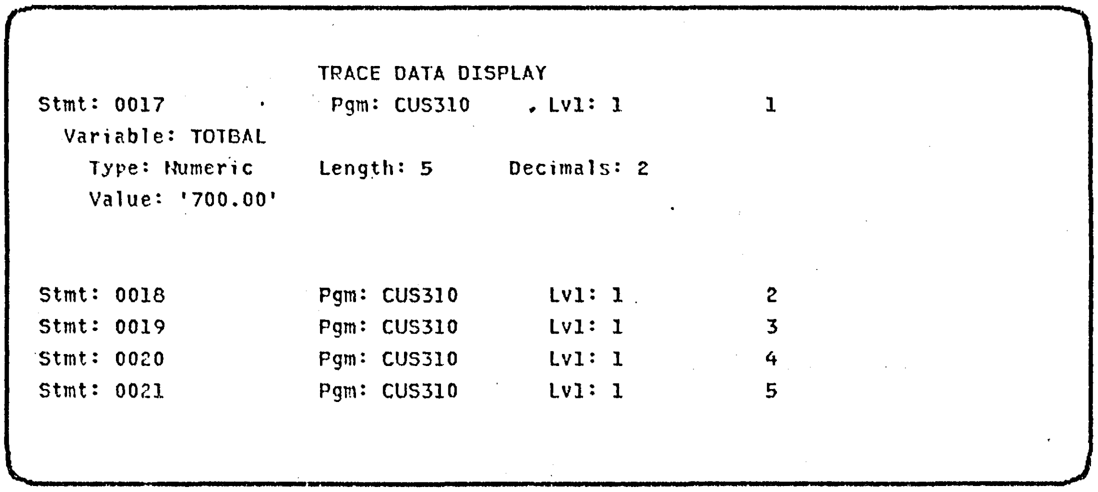
```
ADDTRC      STMT(6 41) PGMVAR(TOTBAL) OUTVAR(*CHG)
```

CUS310 is the default program, so it did not have to be specified.

The specifications for CUS310 look like this:

```
Statement
Numbers   •
  •       •
  •       •
  •
                C
0006            C           START     TAG
0007            C                     EXFMTPROMPT
0008            C   98                SETON                     LR    98=END OF PROG
0009            C   98                RETRN
0010            C   99                GOTO START                      99=HELP KEY
                C
0011            C           CUST      CHAINCUSREC               61    61=NOT FOUND
                C
          •
          •
          •
0016            C* INQUIRY, UPDATE, DELETE REQUESTS
0017            C           ARBAL     ADD  ORDBAL    TOTBAL           TOTAL AMT OWED
0018            C           RESPND    TAG
0019            C                     EXFMTRESPONSE
0020            C   99                GOTO RESPND                     99=HELP KEY
0021            C  N92N93             GOTO START                      INQUIRY
0022            C           CUST      CHAINCUSREC               61    61=NOT FOUND
0023            C   61                GOTO START
                C
          •
          •
          •
                C
0031            C* ADD
0032            C           ADDDSP    TAG
0033            C                     EXFMTADDREC
0034            C   99                GOTO ADDDSP                     99=HELP
0035            C   93                GOTO START                      93=CANCEL
0036            C                     Z-ADD5ØØ.ØØ    CRDLMT           INITIAL CREDIT
0037            C                     WRITECUSREC                     ADD NEW RECORD
                C
0038            C* CONFIRMATION RECORD
0039            C           CONFRM    TAG
0040            C                     WRITECOMPLT
0041            C                     GOTO START
                C
```

The following display results from this trace and is displayed using the Display Trace Data (DSPTRCDTA) command.



```
                      TRACE DATA DISPLAY
Stmt: 0017              Pgm: CUS310        Lvl: 1           1
  Variable: TOTBAL
    Type: Numeric       Length: 5       Decimals: 2
    Value: '700.00'


Stmt: 0018              Pgm: CUS310        Lvl: 1           2
Stmt: 0019              Pgm: CUS310        Lvl: 1           3
Stmt: 0020              Pgm: CUS310        Lvl: 1           4
Stmt: 0021              Pgm: CUS310        Lvl: 1           5
```

On the DSPTRCDTA command you can specify whether the trace information is to be removed from the system or left on the system after the information is displayed. If you leave the trace information on the system, any other traces are added to it. The information remains on the system until the debug job has completed execution. You can also use the Clear Trace Data (CLRTRCDTA) command or the clear option on the Display Trace Data (DSPTRCDTA) command to remove trace information from the system.

Breakpoints can be used within a trace. If a variable changes at a breakpoint and the program resumes execution after the breakpoint, the trace information contains the value of the variable at the time the breakpoint occurred instead of at the time the statement was executed.

DUMPING OBJECTS AND JOBS

To help debug programs you can dump objects and jobs. You can dump the contents of any CPF object stored in a library. The dump is written to a spooled printer file. To dump an object use the Dump Object (DMPOBJ) command and specify the object name and type. When you dump a job you use the Dump Job (DMPJOB) command (it has no parameters). You get a dump of the program activation and invocation stacks, various objects associated with the job, and any temporary work spaces used by the job.

COMMAND LIST

This is a list of commands related to testing. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL*.

Descriptive Name	Command Name	Function
Enter Debug	ENTDBG	Indicates the beginning of debug mode.
End Debug	ENDDBG	Indicates the end of debug mode.
Change Debug	CHGDBG	Changes the debug characteristics of a debug job.
Display Debug	DSPDBG	Displays a list of the programs being debugged and other debug information.

Debug Program

Descriptive Name	Command Name	Function
Add Program	ADDPGM	Adds a program to debug mode.
Remove Program	RMVPGM	Removes a program from debug mode.

Program Variable

Descriptive Name	Command Name	Function
Change Program Variable	CHGPGMVAR	Changes the value of a program variable.
Display Program Variable	DSPPGMVAR	Displays the value of a program variable.

Program Pointer

Descriptive Name	Command Name	Function
Change Pointer	CHGPTR	Changes the value of a pointer.

Breakpoint

Descriptive Name	Command Name	Function
Add Breakpoint	ADDBKP	Adds a breakpoint to a program in debug mode.
Remove Breakpoint	RMVBKP	Removes a breakpoint from a program in debug mode.
Return Breakpoint	RTNBKP	Returns execution from one breakpoint to a previous breakpoint. Used when there is more than one program at a breakpoint.
Resume Breakpoint	RSMBKP	Resumes execution of a program after a breakpoint has been displayed.
Display Breakpoint	DSPBKP	Displays breakpoint information.

Trace

Descriptive Name	Command Name	Function
Add Trace	ADDTRC	Adds a trace to a program in debug mode.
Remove Trace	RMVTRC	Removes a trace from a program in debug mode.
Display Trace	DSPTRC	Displays a list of traces defined for a program.
Clear Trace Data	CLRTRCDTA	Removes data from previous trace operations.
Display Trace Data	DSPTRCDTA	Displays trace information from a previously generated trace.

Dumps

Descriptive Name	Command Name	Function
Dump Job	DMPJOB	Dumps the basic data structure of a job.
Dump Object	DMPOBJ	Dumps a CPF object.

Part 5. Work Management

Chapter 16. Work Management

Work management functions manage jobs on the system, allocate resources for use within jobs, and schedule jobs for execution. You can tailor these functions to meet your installation's needs.

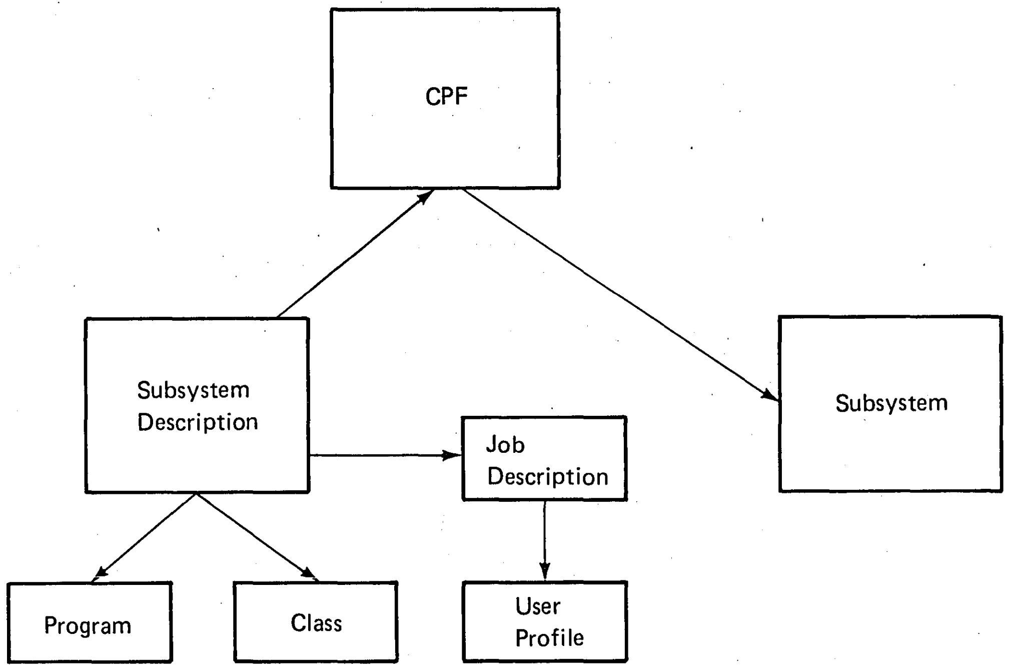
The basic unit of work on System/38 is a job. An *interactive job* is a job in which the processing actions are performed by the system in response to input provided by a work station user. During the job, a dialog exists between the user and the system. For example, between sign-on and sign-off you could create a physical file, delete a display file, and call a program. Each is a single command, but together they are a job.

A *batch job* is a job in which the processing actions are submitted as a predefined series of actions to be performed without a dialog between the user and the system. The job consists of all the processing actions that result from input contained within the job. Basically, a batch job is all the commands between a Job command and an End Job command or the next Job command. For example, the following shows three jobs:

```
//JOB OEDAILY        }  Job: No other commands besides
                           Job command.

//JOB OEWEEKLY       )
 CALL PGMA           }  Job: Commands included in job.
 CALL PGMB           )
 CALL PGMC           )

//JOB OEMONTHLY      )
 CALL PGME           )
 //DATA              )
 .                   }  Job: Inline data file used in job.
 .                   )
 .                   )
//ENDJOB             )
```

An autostart job, a reader, a writer, and a job submitted using the SBMJOB command are also considered batch jobs although there are no JOB commands associated with them.

The following shows the basic objects CPF needs to control a subsystem and run a job:



On System/38, jobs execute in subsystems. A *subsystem* is an operating environment through which CPF coordinates work flow and resource usage. The specifications that define a subsystem and that CPF uses to control the subsystem are contained in an object known as a subsystem description. The *subsystem description* contains routing information necessary for determining what program must be invoked for a job and what the execution parameters are. The subsystem description references another object known as a class. A *class* specifies the execution parameters for a job. (See *Routing Entries* in this chapter for more information about classes.)

Each job in a subsystem must have an associated job description, but more than one job can use the same job description. A *job description* is an object in which the attributes of a job are predefined and stored. The job description references another object known as a user profile. A *user profile* represents a particular user or users to the system. It identifies which objects and functions the user is authorized to use. (See Chapter 18, *Security* for more information about user profiles.)

You can run your jobs using the IBM-supplied objects or objects that you create. IBM supplies objects needed for batch jobs, interactive jobs, and spooling. You can use these objects as they are or tailor them to your installation's needs.

When you get your system, you get the following IBM-supplied subsystem descriptions:

- QCTL1 and QCTL2 (interactive subsystems)
- QBATCH (batch subsystem)
- QSPL (spooling subsystem)

All these subsystem descriptions are in the general purpose library (QGPL).

The IBM-supplied interactive subsystem (QCTL1 or QCTL2) supports all the interactive jobs processed through the display work stations (including the system console) on the system. Either QCTL1, QCTL2, or some other subsystem defined by your installation can be the controlling subsystem. A *controlling subsystem* is the subsystem automatically started when the system is started. There are two copies (QCTL1 and QCTL2) of the IBM-supplied interactive subsystem. You can make changes to the subsystem by changing an inactive copy. The controlling subsystem is specified in the system value QCTLSBSD. The class specified in the routing entries for this subsystem is QCTL.

After QCTL1 or QCTL2 is started you do not have to enter a user password to sign on; the QUSER user profile is the specified default. The routing data is QCMDI, which causes the IBM-supplied control language processor to be invoked for the routing step.

The IBM-supplied batch subsystem (QBATCH) supports all the batch jobs processed on the system. The batch subsystem must be started by a Start Subsystem (STRSBS) command and terminated explicitly by a Terminate Subsystem (TRMSBS) command, a Terminate CPF (TRMCPF) command, or a Power Down System (PWRDWNSYS) command. Batch jobs are placed on the QBATCH job queue for processing. The batch subsystem does not have to be started for jobs to be placed on the queue. When the subsystem is started, the jobs on the queue are processed. The routing data QCMDB causes the routing entry to invoke IBM-supplied control language processor for the routing step. The class specified in the routing entries for this subsystem is QBATCH.

The spooling subsystem (QSPL) supports reading jobs and job streams and writing the output from the jobs. Jobs are placed on a job queue, either QBATCH or a designated job queue, by a reader (a CPF program) and the output is written from a spooled file on the QPRINT output queue to a device by a writer (a CPF program). To submit a job to a job queue using the Submit Job (SBMJOB) command, the spooling subsystem does not have to be active. The class specified in the routing entries for this subsystem is QSPL.

In addition, you get the following IBM-supplied job descriptions:

- QCTL (interactive job)
- QBATCH (batch job)
- QSPLDBR (data base spooling reader)
- QSPLCRDR (card read spooling reader)
- QSPLDKTR (diskette spooling reader)
- QSPLPRTW (printer spooling writer)
- QSPLCRDW (card punch spooling writer)
- QSPLDKTW (diskette spooling writer)

All these job descriptions are in the general purpose library (QGPL).

The most commonly used IBM-supplied user profiles needed to perform jobs are:

- For a batch job, the programmer user profile QPGMR. QPGMR is referenced in the IBM-supplied batch job description QBATCH.
- For an interactive job, the work station user profile QUSER. QUSER is referenced in the IBM-supplied interactive job description QCTL.

Work Entries and Routing Entries

Before you can understand how a subsystem executes your jobs, you must understand work entries and routing entries, which are contained in the subsystem description.

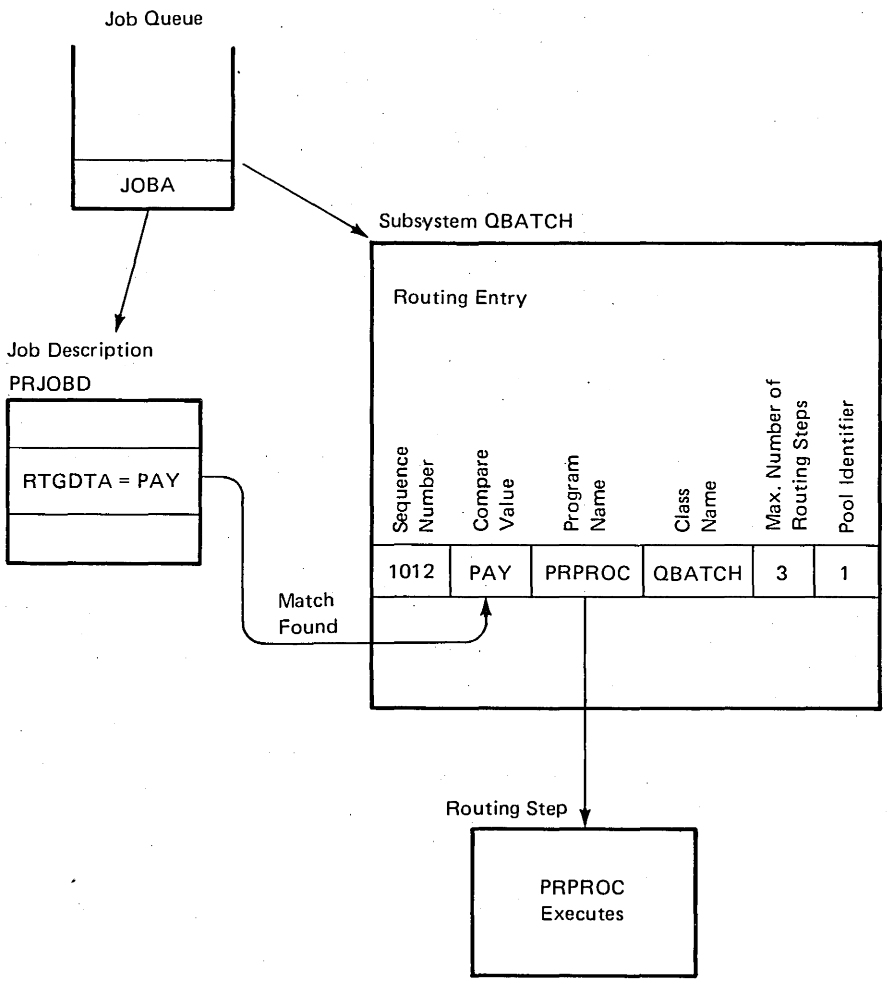
A work entry defines a source of jobs for a subsystem:

- From a work station when a user signs on or when a user transfers into a subsystem (work station entry)
- From a job queue (job queue entry)
- Automatically started when the subsystem is started (autostart job entry)

See *Work Entries* in *Subsystem Descriptions* in this chapter for what needs to be specified to define each type of work entry.

A routing entry specifies what program must be invoked for a job's routing step to execute in the subsystem. Which routing entry is used is determined by comparing routing data for a job to a compare value specified in the routing entry. For batch jobs, routing data comes from the job description or from the RTGDTA parameter of the JOB or SBMJOB command. For interactive jobs, routing data comes from the job description or from a display in which the user enters data. When a match is found, a routing step is initiated. A routing step consists of the execution of the program specified in the routing entry.

The following illustrates the concept of routing entries. The routing data for job JOBA is contained in the job description PRJOBD. The routing data PAY matches the routing entry with sequence number 1012 and the program PRPROC is executed in the subsystem QBATCH.



See *Routing Entries* in *Subsystem Descriptions* in this chapter for what needs to be specified to define a routing entry.

WAYS JOBS ARE INITIATED ON THE SYSTEM

There are a number of ways that jobs can be initiated on System/38. For interactive jobs, those ways are:

- Using routing data from the job description to automatically route jobs. Which programs are executed depends on what work station is being used. The following can be specified for the routing step:
  - The IBM-supplied command processor QCL is specified in the routing entry and no initial program is specified in the user profile. The command entry display is displayed to the user. *No changes are made to the IBM-supplied objects.* (See *Interactive Jobs, Executing QCL for the Routing Step.*)
  - The IBM-supplied command processor QCL is specified in the routing entry but an initial program is specified in the user profile. This initial program can be the IBM-supplied program QCALLMENU or a user program. The initial program is executed for the routing step. *If a user program is specified as the initial program, only the user profile need be changed. For QCALLMENU, no changes need be made.* (See *Interactive Jobs, Executing QCL for the Routing Step.*)
  - A user program is specified in the routing entry and is executed for the routing step. *You must add a routing entry specifying the user program to the subsystem description.* (See *Interactive Jobs, Executing a User Program for the Routing Step.*)

- Using input from a display as routing data. This input can come from an IBM-supplied display or a user-defined display. *You must add a work station entry specifying the device file containing the display format to the subsystem description.* (See *Interactive Jobs, Prompting for Routing Data.*)

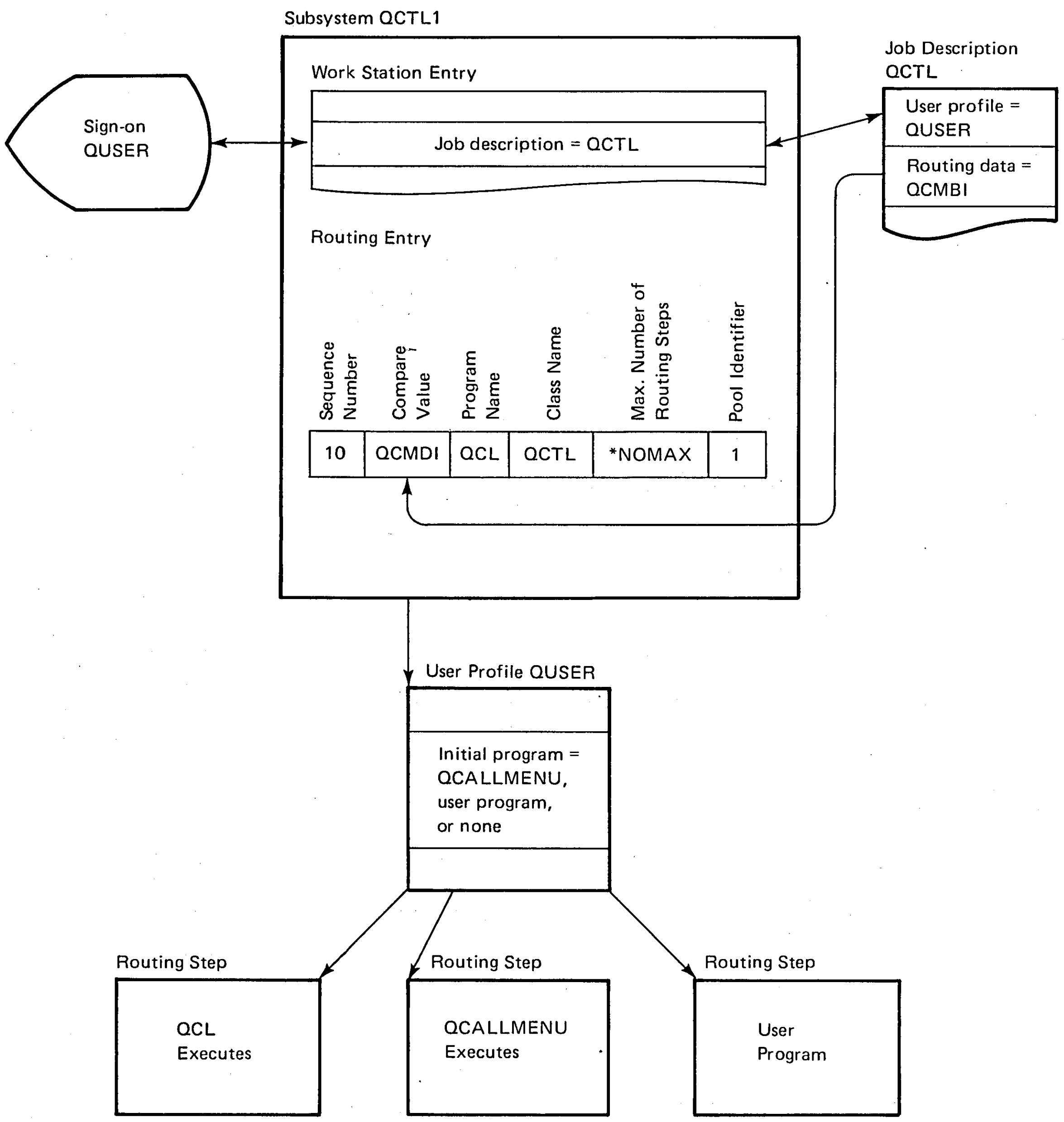
For batch jobs, the ways jobs are initiated are:

- Using a spooling reader or a SBMJOB command to place jobs on job queues. The routing data comes from the job description or JOB or SBMJOB command. *You need not make any changes to the IBM-supplied objects.* (See *Batch Jobs, Queuing Jobs.*)

- Using autostart job entries. A job is automatically started when the subsystem is started. *You must add an autostart job entry and a routing entry to the subsystem description.* (See *Batch Jobs, Using Autostart Jobs.*

Interactive Jobs, Executing QCL for the Routing Step

In the following illustration, the routing data matches a routing entry specifying that QCL be executed for the routing step.

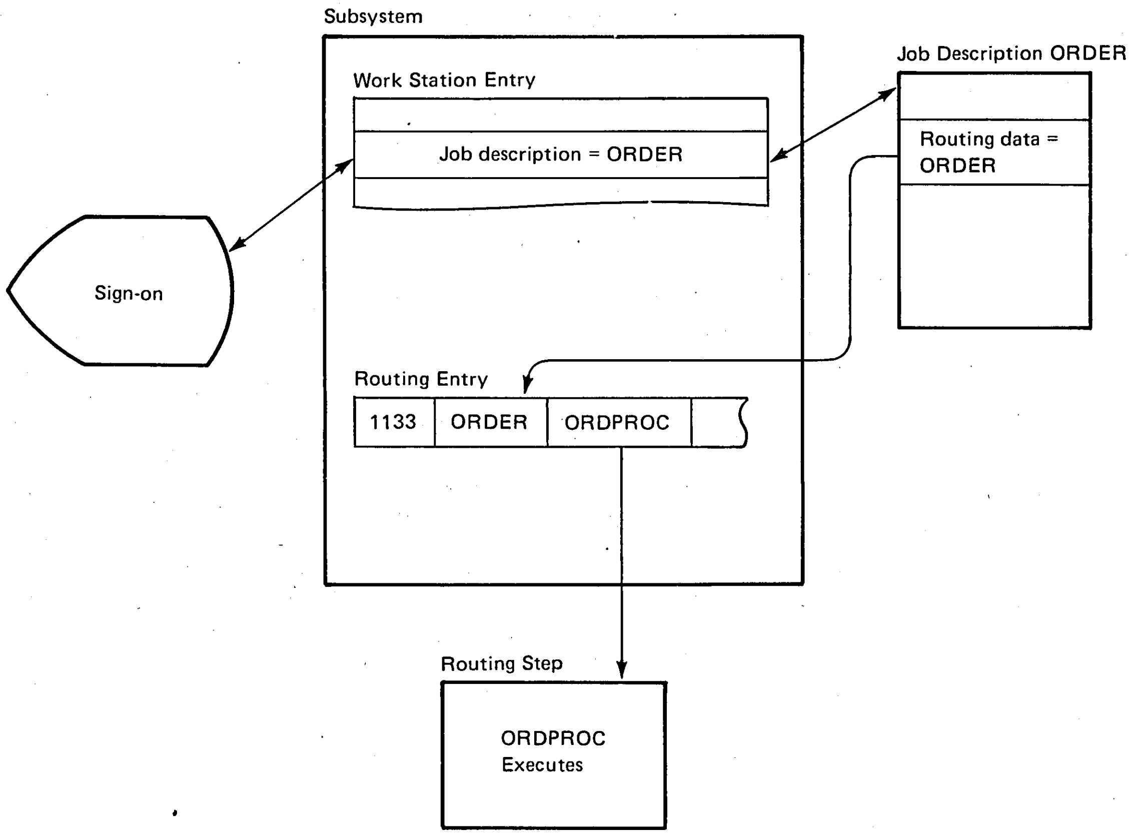


The work station entry contains the job description name for the interactive job. After sign-on, the system searches the routing entries in the subsystem description to find a match for the routing data in the job description. The routing data is QCMDI, and it matches the routing entry that specifies the program QCL. QCL checks the user profile to determine if an initial program has been specified for this user. If none is found, QCL displays the command entry display to the user. If an initial program is found, it is executed. This initial program can be QCALLMENU or an application program.

The initial program can be used to establish the environment in which other commands are executed. For example, the library list can be changed or message files can be overridden so that messages in a non-English language can be used. In addition, the initial program can be used to restrict the user to a specific set of functions. As long as the initial program does not return normally to the program QCL, the command entry display is not displayed and the user cannot enter commands. The IBM-supplied call menu program (QCALLMENU) uses this technique to restrict users to the options of the menu it displays. .

Interactive Jobs, Executing a User Program for the Routing Step

In the following illustration, the routing data matches a routing entry specifying that a user program be executed for a routing step.

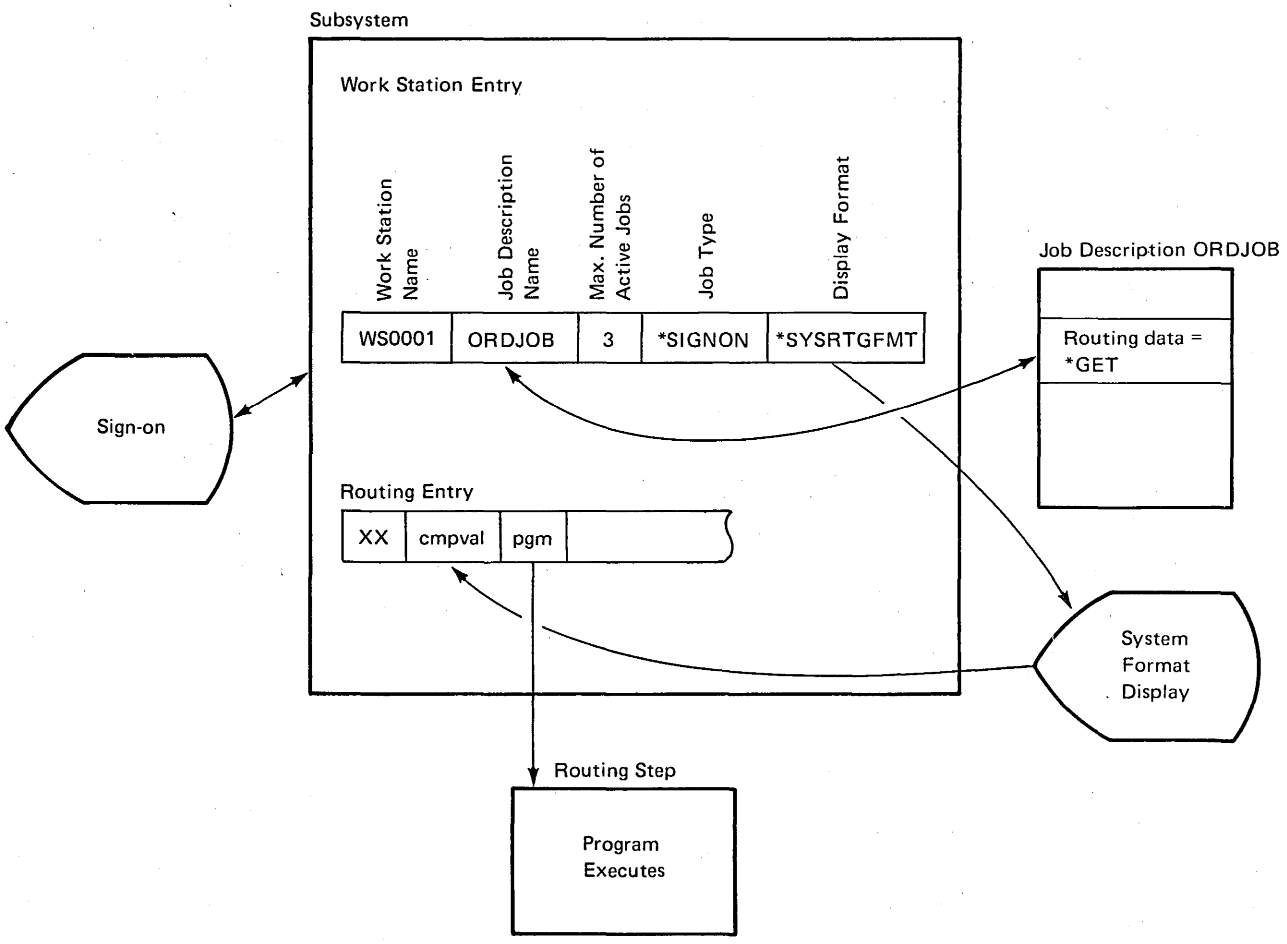


The routing data in the job description is ORDER, and it matches to routing entry 1133, which invokes the user program ORDPROC.

For this type of execution, the routing data in the job description is user-defined. A routing entry must be created by the user. The routing entry can be added to an IBM-supplied subsystem or to a user-defined subsystem description.

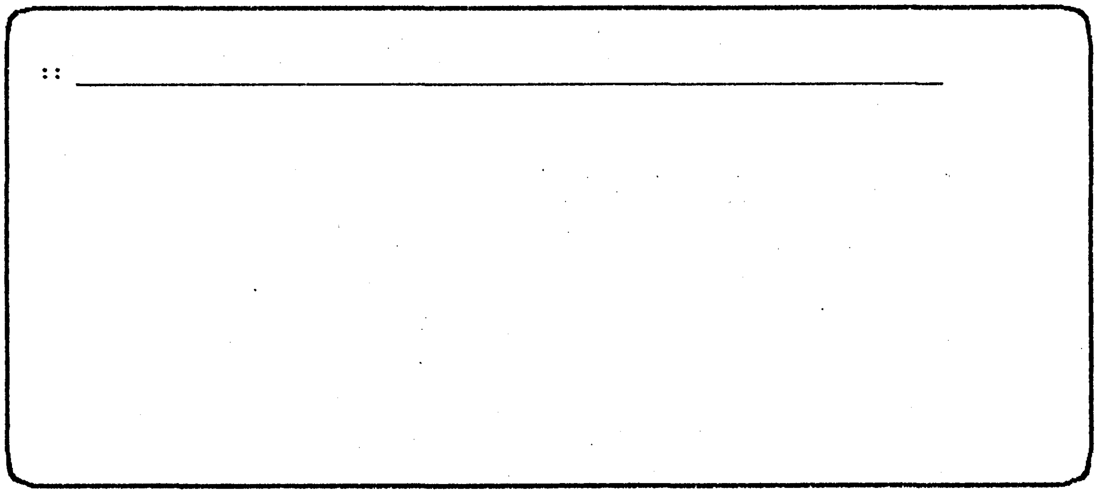
Interactive Jobs, Prompting for Routing Data

In the following illustration, the routing data in the job description specifies that the routing data must be prompted for.



When the subsystem checks for routing data in the job description, it finds *GET. The work station entry contains the name of the record format for a display and the name of the device file containing the record format. This display is a prompt that requests routing data. The display can be an IBM-supplied display or a user-defined display.

In the case of the preceding illustration, the IBM-supplied display is used (indicated by *SYSRTGFMT):



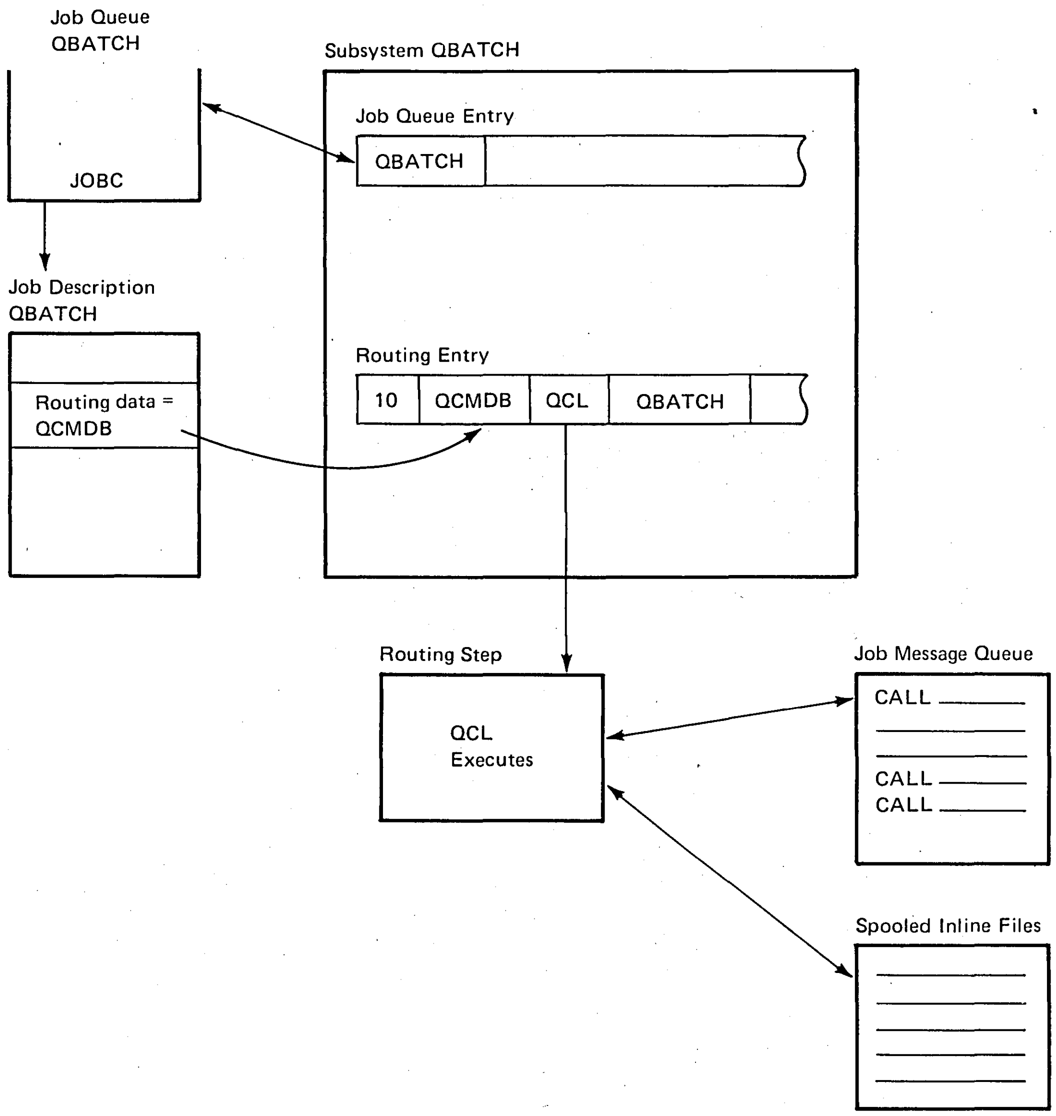
The data entered by a user goes into a record called the data management feedback area. The routing data from this feedback area used by the subsystem contains 80 characters:

- Ten-character device name
- Two-character command key identifier
- First 68 characters of data entered on the display

For a user-defined display, you must define the routing data format in the file description for the device file using the routing field DDS keywords.

Batch Jobs, Queuing Jobs

The following illustrates how jobs on a job queue are initiated.



Jobs are placed on a job queue:

- When a reader (a CPF program) processes a JOB command
- Through a SBMJOB command
- Through a TFRJOB command

For discussions of submitting a job and transferring a job, see *Submitting a Job from Another Job* and *Rerouting and Transferring Jobs*, respectively.

A reader reads jobs from cards, diskettes, or a data base file and places the jobs on a job queue. A job can contain request data and inline data files. The request data is placed on a job message queue. Any inline data files are spooled for access when the job executes. Normally, the request data for a job is one or more commands.

The location of the request data is determined by the RQSDTA parameter on the JOB, SBMJOB, or TFRJOB command. The RQSDTA parameter can specify any one of the following:

- The actual request data
- That the request data is in the job description
- That the request data follows the JOB command

When QCL executes, it retrieves the request data (commands) from the job message queue and executes it, using spooled inline data files if any exist for the job.

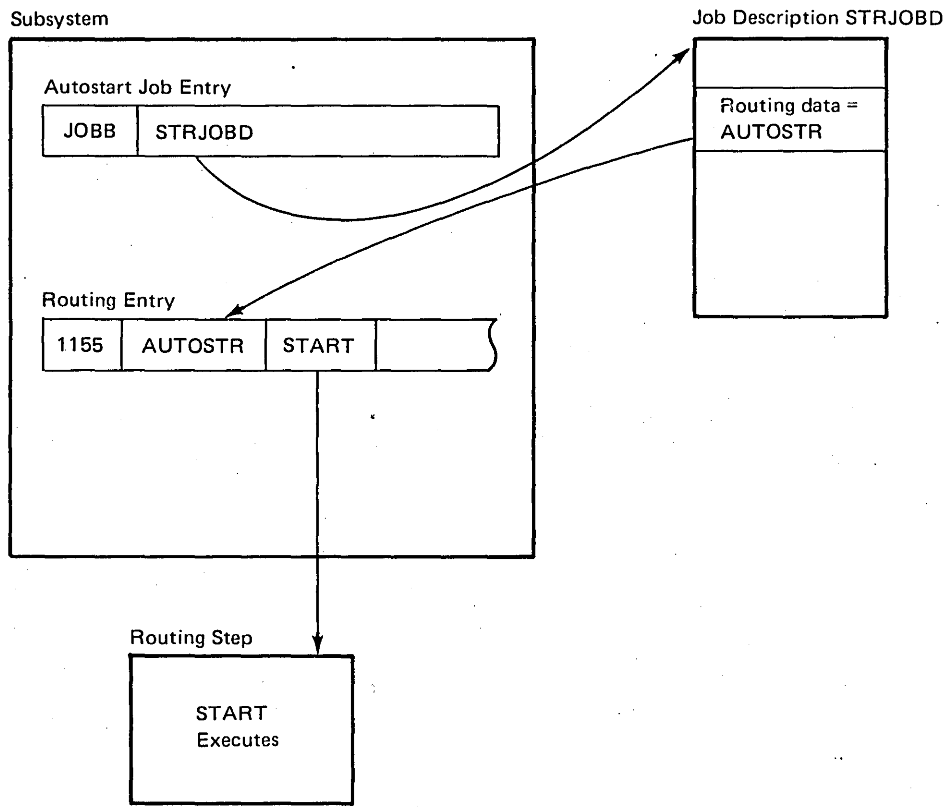
When a subsystem is started, a job queue entry in the subsystem description tells the subsystem what queue to look on for jobs to process. There can only be one job queue entry in a subsystem description. More than one subsystem description can reference a single job queue but only one subsystem description can use the job queue at a time.

The routing data used to route a job can come from the job description or from a JOB, SBMJOB, RRTJOB, or TFRJOB command.

In the preceding illustration the routing data QCMDB comes from the IBM-supplied job description QBATCH. The routing data matches the IBM-supplied routing entry 10, and QCL is executed.

Batch Jobs, Using Autostart Jobs

The following illustrates how autostart jobs are executed.



An autostart job is automatically started when the subsystem is started. For each autostart job, an autostart job entry must be added to a subsystem description. All that is specified in the routing entry is the job name and job description name. The subsystem uses the routing data specified in the job description to find the program to be executed.

SUBMITTING A JOB FROM ANOTHER JOB

You can submit a job from within an executing job (for example, submit a batch job from a display work station even though you are in an interactive subsystem). The submitted job is placed on a job queue.

To submit a job in this manner, use the Submit Job (SBMJOB) command. Submitting a job consists of specifying a job description name and, if necessary, overriding the parameters in the job description for the job. A job name can be specified but if it is not, it is assumed to be the same as the simple job description name.

REROUTING AND TRANSFERRING JOBS

Rerouting jobs is similar to transferring jobs except that rerouting initiates a new routing step within the same subsystem while transferring initiates a new routing step in another subsystem. In both cases, file overrides are removed, allocated objects except the work station and job message queues are deallocated, and files are closed. The library list stays the same because it is specified at the job level. The temporary library QTEMP is not deleted, and all job attributes remain the same.

If you are signed on and want to get to another subsystem, you can transfer your job to another subsystem. However, to transfer a job you must have operational rights to both the job queue and the subsystem description of the subsystem being transferred to.

To transfer an interactive job, the subsystem being transferred to must be active. When the interactive job is placed on the job queue it is given the highest job queue priority to prevent its being held too long on the job queue.

Rerouting a job is a method of changing processing attributes for a job. By rerouting a job, you can invoke a different program to process the job. For example, a job is executing. The next part of the job needs to execute in a different storage pool. You reroute the job at this time to execute the job in a different storage pool.

POOLS

Routing steps execute within storage pools. A pool is a logical division of main storage. Pools are used to reduce contentions between programs for main storage. For example, a high-performance application executes in a pool separate from other applications. Routing steps executing in the same pool only contend with each other for main storage requirements; they only contend with routing steps executing within other pools for resources other than main storage.

The attributes of a storage pool are its size and its activity level. Its size is measured in K bytes. The activity level of a pool indicates how many routing steps can execute in the pool at the same time. Excess jobs are automatically queued.

IBM-supplied subsystem descriptions contain defined pools. You might want to change the size of these pools to tailor them to your needs.

In addition to the pools defined in the subsystem descriptions, there are two other pools on the system:

- A base pool (*BASE) that can be shared between subsystems. This pool contains the unassigned storage on the system plus the amount specified in the system value QBASPOOL. This is the pool for the IBM-supplied batch subsystem (QBATCH).

- A machine pool used by the machine for its processing. The storage for this pool is specified in the system value QMCHPOOL.

You could set up the storage pools in either of the following ways:

1. Use the base pool as the pool for an interactive subsystem, set up a pool for a batch subsystem, and keep the machine and spooling pools as they are.

2. Use the base pool and two or three application pools.

There is a maximum of 16 pools on the system, two of which are the base and machine pools. Therefore, you can define as many as 14 pools for active subsystems, but only 10 for a subsystem.

ACTIVITY LEVELS

An activity level defines how many jobs can be active concurrently within an area such as a subsystem or pool. Activity levels can be specified for:

- *System.* The maximum number of jobs, initially 100 jobs, that can be active concurrently in the system. You can change this by changing the system value QMAXACTLVL.

- *Subsystem.* The maximum number of jobs that can be active concurrently in the system. The maximum can be changed using the Change Subsystem Description (CHGSBSD) command.

- *Job queue entry.* The maximum number of jobs that can be active concurrently for a job queue entry. The maximum can be changed using the Change Job Queue Entry (CHGJOBQE) command.

- *Work station type entry.* The maximum number of jobs that exist for the work station type entry. The maximum can be changed using the Change Work Station Entry (CHGWSE) command.

- *Routing entry.* The maximum number of routing steps that can be active concurrently for the routing entry. The maximum can be changed using the Change Routing Entry (CHGRTGE) command.

- *Pool.* The maximum number of routing steps that can execute concurrently in a pool. The maximum can be changed using the Change Subsystem Description (CHGSBSD) command.

SUBSYSTEM DESCRIPTIONS

A subsystem description consists of three parts:

- Subsystem attributes
  - Storage pool definitions (sizes and activity levels)
  - Maximum number of jobs that can be active in a subsystem concurrently
  - Public authority
  - Text description
- Routing entries
- Work entries

The subsystem attributes are supplied through the Create Subsystem Description (CRTSBSD) command. The routing entries are added to the description as a result of the Add Routing Entry (ADDRTGE) command. The work entries are added as a result of the following commands:

- Add Autostart Job Entry (ADDAJE)
- Add Work Station Entry (ADDWSE)
- Add Job Queue Entry (ADDJOBQE)

You can change the IBM-supplied subsystem descriptions or any user-created subsystem descriptions by using the following commands.

- Add Autostart Job Entry (ADDAJE)
- Add Job Queue Entry (ADDJOBQE)
- Add Work Station Entry (ADDWSE)
- Add Routing Entry (ADDRTGE)
- Change Subsystem Description (CHGSBSD)
- Change Autostart Job Entry (CHGAJE)
- Change Job Queue Entry (CHGJOBQE)
- Change Work Station Entry (CHGWSE)
- Change Routing Entry (CHGRTGE)
- Remove Autostart Job Entry (RMVAJE)
- Remove Job Queue Entry (RMVJOBQE)
- Remove Work Station Entry (RMVWSE)
- Remove Routing Entry (RMVRTGE)

Routing Entries

A routing entry defines how a routing step is to be initiated. A routing entry contains the following:

- A routing entry sequence number
- A routing data compare value and starting position for the comparison
- The name of the program to be invoked
- The name of the class to be used for the routing step
- The maximum number of routing steps that can be active concurrently for the entry
- The identifier of the pool in which the routing step is to execute

When you add a routing entry to a subsystem description, you assign a sequence number to the entry. This sequence number tells the subsystem the order in which routing entries are to be searched for a routing data match. For example, you have five routing entries for the subsystem description IOESBSD (the order in which they were added to the subsystem is the order in which they are listed here).

Sequence Number	Compare Value
0010	1
0030	2
0040	3
0020	5
0050	4

These routing entries are searched in sequence number order. If the routing data was 2, the search ends with routing entry 0030.

You can specify a compare value (*ANY) on the highest numbered routing entry. *ANY means that a match is forced.

In addition to indicating which program is to be invoked, the routing entry specifies which class is to be used to execute the routing step. A class contains parameters to control execution in a routing step:

- Machine execution priority. The priority the routing step has when competing with other routing steps for machine resources.
- Time slice. A quantity of processor time allowed for the routing step before other waiting routing steps are given the opportunity to execute.

- Purge. Whether a job is to be moved from main storage to auxiliary storage either at the end of a time slice or when entering a long wait.

- Default maximum instruction wait time. The maximum time an instruction is to wait for completion.

- Maximum CPU time. The maximum time the processor can execute a routing step before the routing step is canceled.

- Maximum temporary storage. The maximum amount of temporary auxiliary storage that a routing step can use before the routing step is canceled.

The default maximum instruction wait time, maximum CPU time, and maximum temporary storage can help stop an erroneous program from impairing system performance.

Work Entries

Work entries identify the sources for jobs that are processed in a subsystem. There are three types of work entries:

- *Autostart*. The job is automatically started when the subsystem is started.

- *Job queue*. Jobs to be processed are taken from the specified job queue.

- *Work station.* The job processed when a work station user signs on or when he transfers a job from another subsystem.

The descriptions of these work entries (which are part of the subsystem description) are:

- For an autostart job entry:
  - Job name
  - Job description name

- For a job queue entry:
  - Job queue name
  - Maximum number of jobs that can be active concurrently from the queue

- For a work station entry:
  - Work station name or type
  - Job description name
  - Maximum number of jobs that can exist for the entry
  - Job type (demand sign-on or entry)
  - Format of the display used to obtain the routing data if the routing data is not specified in the job description

Note: A demand sign-on job type results from a user signing on at a display work station. An entry job is an interactive job transferring to a subsystem from another subsystem.

Creating a Batch Subsystem Description

The following is an example of creating a batch subsystem description. This example shows creating the subsystem description, creating a job queue and adding it to the subsystem description, and adding a routing entry to the subsystem description.

NIGHTQ is the subsystem description for the concurrent nighttime jobs. One pool (pool 1) is specified for the subsystem and is defined as the base pool (*BASE). There is no maximum on the number of jobs that can execute concurrently in the subsystem. The following Create Subsystem Description (CRTSBSD) command creates NIGHTQ:

```
CRTSBSD  SBSD(NIGHTQ) POOLS((1 *BASE))
         TEXT('Concurrent nighttime jobs')
```

NIGHTQ is placed in the general purpose library (QGPL) by default.

The job queue for the subsystem description NIGHTQ is also named NIGHTQ and, by default, is placed in the library QGPL. The following Create Job Queue (CRTJOBQ) command creates the NIGHTQ queue:

```
CRTJOBQ  JOBQ(NIGHTQ)
         TEXT('Concurrent nighttime job queue')
```

Any user profile having job control authority can control the jobs on the job queue. Also, you can control your own jobs.

The NIGHTQ job queue is defined to the subsystem through a job queue entry. The following Add Job Queue Entry (ADDJOBQE) command adds the job queue entry for the NIGHTQ job queue to the subsystem description NIGHTQ:

```
ADDJOBQE SBSD(NIGHTQ) JOBQ(NIGHTQ) MAXACT(3)
```

For this entry, three jobs from the queue can be active concurrently within the subsystem.

The routing entry for the NIGHTQ subsystem description indicates that the IBM-supplied control language processor QCL (in library QSYS) is to be invoked to process the routing step, that the routing step is to execute in pool 1 (which in this case is the base pool), and that the class QCLASSB is to be used to execute the routing step. There is no maximum on the number of routing steps that can be active for this routing entry. The following Add Routing Entry (ADDRTGE) command adds the routing entry to the NIGHTQ subsystem description:

```
ADDRTGE  SBSD(NIGHTQ) SEQNBR(10)
         CMPVAL (QCMDB) PGM(QCL.QSYS)
         CLS(QCLASSB) POOLID(1)
```

Creating an Interactive Subsystem Description

The following is an example of creating an interactive subsystem description that automatically starts up and shuts down a group of display work stations. This example shows creating the subsystem description and adding a work station entry to the description.

ORDER is the subsystem description for order entry jobs. Two pools (pools 1 and 2) are defined for the subsystem. Pool 1 is to contain 60K bytes of storage and have an activity level of 1; pool 2 is to contain 150K bytes of storage and have an activity level of 3. The following CRTSBSD command creates ORDER.

```
CRTSBSD  SBSD(ORDER) POOLS((1 60 1) (2 150 3))
         TEXT('Order entry jobs')
```

The work station entry for the ORDER subsystem description specifies that all work stations that are 5251 Display Stations are to be allocated to the subsystem when the subsystem is started. Likewise, the work stations are deallocated when the subsystem is terminated. By default, the sign-on prompt is displayed on the allocated work stations. The following Add Work Station Entry (ADDWSE) command adds this entry to the subsystem description:

```
ADDWSE SBSD(ORDER) WRKSTNTYPE(5251) JOBD(ORDER)
```

JOB DESCRIPTIONS

A job description can contain the following job attributes:

- *User name.* What user profile the job is to use.
- *Job queue.* The job queue on which the job is placed (batch job only).
- *Job priority.* Priority of the job on the job queue (batch job only).
- *Output priority.* Priority of the spooled output files produced by the job (batch job only).
- *Routing data.* Data used to determine which routing entry to use for the job.
- *Request data.* Data to be put in the job message queue and passed to the program that is invoked.
- *Syntax checking.* Whether the reader is to syntactically check the input and at what message severity (for syntax errors) the system should prevent processing of the job (batch job only).
- *Initial library list.* What the user part of the initial library list for the job is.
- *Cancel severity.* At what message severity (for job execution) the job should stop processing (batch job only).

- *What to log in the job log.* The level of information to be sent to the job log, the messages to be sent to the job log (determined by a severity code), and the level (first-level or first- and second-level) of message text to be sent to the job log.

- *Output queue.* The default output queue on which the spooled output files produced by the job are to be placed (batch job only).

- *Hold on queue.* Whether a job being placed on the job queue is to be held on the job queue until it is released (batch job only).

- *Job date.* The date to be used for the job (if different from the system date).

- *Job switches.* The switch settings to be used for the job.

When you submit a job you can use the job description in one of two ways:

- Use a specified job description without changing it for a specified job. For example,

  ```
  //JOB JOBD(QBATCH) JOB(OEDAILY)
  ```

- Use a specified job description but override some of the attributes (using the JOB or SBMJOB command) for a specified job. For example, to change the cancel severity for a job you enter:

  ```
  //JOB JOBD(QBATCH) JOB(OEDAILY) CNLSEV(35)
  ```

Note: For autostart or work station jobs, the job description cannot be overridden.

COMMAND LIST

This is a list of commands related to jobs, subsystem descriptions, classes, and queues. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL*.

Subsystem Descriptions and Classes

Subsystem Descriptions

Descriptive Name	Command Name	Function
Create Subsystem Description	CRTSBSD	Creates a subsystem description.
Delete Subsystem Description	DLTSBSD	Deletes a subsystem description.
Change Subsystem Description	CHGSBSD	Changes the attributes of a subsystem description.
Display Subsystem Description	DSPSBSD	Displays a subsystem description.

Subsystem Autostart Job Entries

Descriptive Name	Command Name	Function
Add Autostart Job Entry	ADDAJE	Adds an autostart job entry to a subsystem description.
Remove Autostart Job Entry	RMVAJE	Removes an autostart job entry from a subsystem description.
Change Autostart Job Entry	CHGAJE	Changes an autostart job entry.

Subsystem Work Station Entries

Descriptive Name	Command Name	Function
Add Work Station Entry	ADDWSE	Adds a work station entry to a subsystem description.
Remove Work Station Entry	RMVWSE	Removes a work station entry from a subsystem description.
Change Work Station Entry	CHGWSE	Changes a work entry.

Subsystem Job Queue Entries

Descriptive Name	Command Name	Function
Add Job Queue Entry	ADDJOBQE	Adds a job queue entry to a subsystem description.
Remove Job Queue Entry	RMVJOBQE	Removes a job entry from a subsystem description.
Change Job Queue Entry	CHGJOBQE	Changes a job queue entry.

Subsystem Routing Entries

Descriptive Name	Command Name	Function
Add Routing Entry	ADDRTGE	Adds a routing entry to a subsystem description.
Remove Routing Entry	RMVRTGE	Removes a routing entry from a subsystem description.
Change Routing Entry	CHGRTGE	Changes a routing entry.

Job Descriptions

Descriptive Name	Command Name	Function
Create Job Description	CRTJOBD	Creates a job description.
Delete Job Description	DLTJOBD	Deletes a job description.
Display Job Description	DSPJOBD	Displays a job description.

Classes

Descriptive Name	Command Name	Function
Creates Class	CRTCLS	Creates a class.
Delete Class	DLTCLS	Deletes a class.
Display Class	DSPCLS	Displays the contents of a class.

System and Job Control

System

Descriptive Name	Command Name	Function
Terminate CPF	TRMCPF	Puts CPF in service only mode.
Power Down System	PWRDWNSYS	Direct the system to power down.
Display System	DSPSYS	Displays the status of each subsystem and system job in the system.
Display System Status	DSPSYSSTS	Displays the status of the system (such as storage allocation).

Subsystems

Descriptive Name	Command Name	Function
Start Subsystem	STRSBS	Starts a subsystem for processing.
Terminate Subsystem	TRMSBS	Terminates a subsystem.
Display Subsystem	DSPSBS	Displays information about each job being processed by a subsystem.

Jobs

Descriptive Name	Command Name	Function
Job	//JOB	Identifies the beginning of a batch job.
End Job	//ENDJOB	Identifies the end of a batch job.
Submit Job	SBMJOB	Submits a job to a job queue.
Cancel Job	CNLJOB	Cancels a job and removes it from the system.
Change Job	CHGJOB	Changes the attributes of a job.
Hold Job	HLDJOB	Holds a job from being processed.
Release Job	RLSJOB	Releases a held job.
Reroute Job	RRTJOB	Reroutes a job to within the current subsystem.
Transfer Job	TFRJOB	Transfers a job to another subsystem.
Display Job	DSPJOB	Displays information about a job.

System Values

Descriptive Name	Command Name	Function
Change System Value	CHGSYSVAL	Changes the value of a system value.
Display System Value	DSPSYSVAL	Displays a system value and its attributes.

Spooling

Job Queues

Descriptive Name	Command Name	Function
Create Job Queue	CRTJOBQ	Creates a job Queue.
Delete Job Queue	DLTJOBQ	Deletes a job queue.
Clear Job Queue	CLRJOBQ	Removes jobs from a job queue without deleting the queue.
Hold Job Queue	HLDJOBQ	Holds the entries on a job queue from being processed.
Release Job Queue	RLSJOBQ	Releases a held job queue.
Display Job Queue	DSPJOBQ	Displays the status of jobs on a job queue.

Output Queues

Descriptive Name	Command Name	Function
Create Output Queue	CRTOUTQ	Creates an output queue.
Delete Output Queue	DLTOUTQ	Deletes an output queue.
Change Output Queue	CHGOUTQ	Changes the attributes of an output queue.
Clear Output Queue	CLROUTQ	Removes files from an output queue without deleting the queue.
Hold Output Queue	HLDOUTQ	Holds the entries on an output queue from being processed.
Release Output Queue	RLSOUTQ	Releases a held output queue.
Display Output Queue	DSPOUTQ	Displays the status of files on an output queue.

Spooled Files

Descriptive Name	Command Name	Function
Change Spooled File Attributes	CHGSPLFA	Changes the attributes of a spooled output file.
Hold Spooled File	HLDSPLF	Holds a spooled file from being written.
Display Spooled File	DSPSPLF	Displays the data records in a spooled file.

Readers

Descriptive Name	Command Name	Function
Start Card Reader	STRCRDRDR	Starts a reader to a card device.
Start Data Base Reader	STRDBRDR	Starts a reader to a data base file.
Start Diskette Reader	STRDKTRDR	Starts a reader to a diskette device.
Cancel Reader	CNLRDR	Cancels a reader.
Hold Reader	HLDRDR	Holds a reader.
Release Reader	RLSRDR	Releases a held reader.

Writers

Descriptive Name	Command Name	Function
Start Card Writer	STRCRDWTR	Starts a writer to a card device.
Start Diskette Writer	STRDKTWTR	Starts a writer to a diskette device.
Start Print Writer	STRPRTWTR	Starts a writer to a printer.
Cancel Writer	CNLWTR	Cancels a writer.
Hold Writer	HLDWRT	Holds a writer.
Release Writer	RLSWTR	Releases a held writer.

Other Commands

This is a list of commands that are also related to work management but are not part of the functions presented in this chapter.

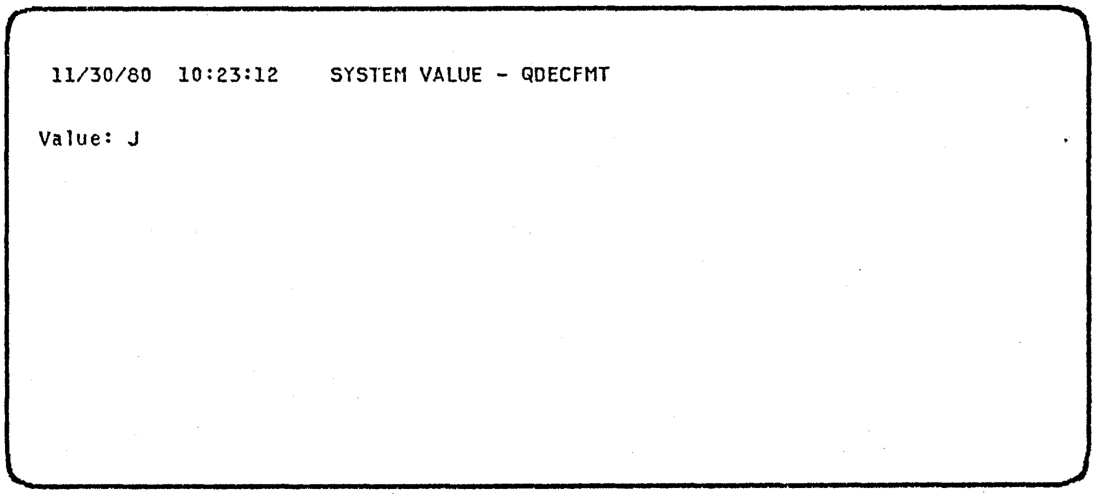
Descriptive Name	Command Name	Function
Display Log	DSPLOG	Chapter 13, Message Handling

Part 6. System Management

Chapter 17. System Values

System values specify attributes of the system such as system date. System values are not objects and cannot be passed as parameter values like CL variables. However, system values can be retrieved and placed in a CL variable as long as the retrieving is performed by a CL program.

System values can be displayed and changed using the Display System Value (DSPSYSVAL) and Change System Value (CHGSYSVAL) commands. The following is an example of a displayed system value.



```
 11/30/80  10:23:12    SYSTEM VALUE - QDECFMT

Value: J
```

The following CHGSYSVAL command changes the value of QUSRLIBL. This system value defines the system default for the user portion of the initial library list. The change does not take effect until after the next CPF startup.

```
CHGSYSVAL SYSVAL(QUSRLIBL) VALUE(DSTPRODLB QGPL QTEMP)
```

You added the library DSTPRODLB to your initial library list for your jobs and placed DSTPRODLB before QGPL and QTEMP.

A system value can be placed in a CL variable for use in a CL program. The Retrieve System Value (RTVSYSVAL) command places the value in a CL variable. (See Chapter 4, *Control Language Programs* for more information.)

System values are provided by IBM. You cannot create them. The following is a list of system values.

QTIME. Time of day. Its value is set when CPF is started. QTIME is a six-character value composed of the following two-character values (each of which can be referenced).

- *QHOUR.* Hour of the day. Its value can range from 00 through 23.
- *QMINUTE.* Minute of the hour. Its value can range from 00 through 59.
- *QSECOND.* Second of the minute. Its value can range from 00 through 59.

QDATE. System date. Its value is set when CPF is started. QDATE is a six-character value (five characters for Julian) composed of the following two-character values (each of which can be referenced).

- *QYEAR.* Year. Its value can range from 0 through 99.
- *QMONTH.* Month of the year. Its value can range from 1 through 12. (Not valid for Julian.)
- *QDAY.* Day of the month. Its value must be a valid day of the specified month. (001 through 356 for Julian.)

The format of the date is as specified in the system value QDATFMT.

QDATFMT. System date format. QDATFMT is three characters whose value can be YMD, MDY, DMY, or JUL (Julian format). (Y = year; M = month; D = day.) This is the format in which the date is displayed.

QHSTLOGSZ. Maximum size of each version of the history log. QHSTLOGSZ is decimal and is initialized to 5000 records.

QSRVLOGSZ. Maximum size of each version of the service log. QSRVLOGSZ is decimal and is initialized to 5000 records.

QMAXACTLVL. Maximum activity level for the system. This is the number of jobs that can execute concurrently. QMAXACTLVL is decimal and is initialized to 100.

QDBRCDWT. Default wait time for obtaining a lock on a data base record. QDBRCDWT is numeric (no decimal positions allowed), is specified in milliseconds, and is initialized to zeros.

QHSTUPDF. Number of history log messages to be enqueued to the history log queue before the history file is updated. QHSTUPDF is numeric and is initialized to 10.

QSRVUPDF. Number of service log messages to be enqueued to the service log queue before the service log is updated. QSRVUPDF is numeric and is initialized to 10.

QSCPFCONS. Start CPF console indicator. Indicates whether Start CPF is to continue unattended or terminate when a console failure occurs during Start CPF. QSCPFCONS is one character that can be '0' (terminate) or '1' (continue unattended). It is initialized to '1'.

QBADPGFRM. Maximum number of bad page frames allowed before Start CPF is terminated. QBADPGFRM is numeric and is initialized to five.

QDATSEP. Character separator for dates displayed by the system. QDATSEP is any one-character value. It is initialized to / (slash).

QDECFMT. Decimal format. It is one of the following characters:

- ƀ: Use a period for a decimal point and zero suppress to the left of the decimal point. QDECFMT is initialized to ƀ.
- J: Use a comma for a decimal point and zero suppress in the second position to the left of the decimal point.
- I: Use a comma for a decimal point and zero suppress to the left of the decimal point.

QSCPFSIGN. Maximum number of Start CPF sign-on attempts allowed. QSCPFSIGN is numeric and is initialized to 15.

QDBRCVYWT. Wait for data base recovery operations during start CPF indicator. QDBRCVYWT is one character that can be '0' (no wait) or '1' (wait). It is initialized to '0'.

QTOTJOB. Initial number of jobs that storage is allocated for at start CPF. The number of jobs is the number supported by the system at any one time, which includes the jobs on the job queue, active jobs, and jobs having output on an output queue. QTOTJOB is numeric and is initialized to 100.

QADLTOTJ. Additional number of jobs that storage can be allocated for during system operation. The storage is allocated after the initial number of jobs (QTOTJOB) is reached. QADLTOTJ is numeric and is initialized to 20.

QCTLSBSD. Controlling subsystem for start CPF. The value of QCTLSBSD is a list of two 10-character values in which the first is the subsystem description name and the second is the library name. QCTLSBSD is initialized to the subsystem QCTL1 in the library QSYS.

QSYSLIBL. Default for system portion of library list. Its value is a list of libraries with each library a 10-character value. The list can contain as many as five names. QSYSLIBL is initialized to QSYS.

QUSRLIBL. Default for user portion of library list. Its value is a list of libraries with each library a 10-character value. The list can contain as many as 10 names. QUSRLIBL is initialized to QGPL and QTEMP in that order.

QBASPOOL. Minimum size of base storage pool in K bytes. QBASPOOL is numeric and is initialized to 150 K bytes.

QMCHPOOL. Initial size of machine storage pool in K bytes. QMCHPOOL is numeric and is initialized to 175 K bytes.

QBASACTLVL. Default activity level for base storage pool. QBASACTLVL is numeric and is initialized to five.

QABNORMSW. System running after abnormal or normal termination indicator. It is one character that can be '0' (normal) or '1' (abnormal). It is initialized to '0'. You cannot change QABNORMSW.

QACTJOB. Initial number of active jobs that storage is allocated for at start CPF. An active job is a job that has been initiated but not terminated or canceled. QACTJOB is numeric and is initialized to 50.

QADLACTJ. Additional number of active jobs that storage can be allocated for during system operation. The storage is allocated after the initial number of active jobs (QACTJOB) is reached. QADLACTJ is numeric and is initialized to 10.

QAUTOIMPL. Start CPF after auto-IMPL indicator. It is one character that can be '0' (no auto-IMPL) or '1' (auto-IMPL). QAUTOIMPL is initialized to 0. You cannot change QAUTOIMPL.

QJOBSPLA. Initial size of the spooling control block for the job. The spooling control block tracks inline spooled files and output spooled files. This size should be specified in multiples of storage increments (page size). QJOBSPLA is numeric and is initialized to 1024 bytes.

QADLSPLA. Additional storage to extend the spooling control block. QADLSPLA is numeric and is initialized to 1024 bytes.

QSPLOUTF1. Number of containers for spooled inline files with record lengths of 96. QSPLOUTF1 is numeric and is initialized to 10.

QSPLOUTF2. Number of containers for spooled inline files of file type source and record lengths of 108. QSPLOUTF2 is numeric and is initialized to 5.

QSPLOUTF3. Number of containers for spooled inline files with record lengths of 128. QSPLOUTF3 is numeric and is initialized to 10.

QSPLOUTF4. Number of containers for spooled inline files of file type source and record lengths of 140. QSPLOUTF4 is numeric and is initialized to 5.

QSPLOUTF5. Number of containers for spooled inline files with record lengths of 256. QSPLOUTF5 is numeric and is initialized to 10.

QSPLOUTF6. Number of containers for spooled inline files of file type source and record lengths of 268. QSPLOUTF6 is numeric and is initialized to 5.

QSPLOUTF7. Number of containers for spooled output files. QSPLOUTF7 is numeric and is initialized to 40.

QAUXSTGTH. Threshold, in terms of a percentage, of unallocated auxiliary storage. QAUXSTGTH is numeric and is initialized to 10.

QSRVONLY. Service mode and nonservice mode indicator. It is one character that can be '0' (standard CPF) or '1' (service only). QSRVONLY is character and is initialized to '0'.

QJOBMSGQSZ. Initial size of the job message queue. QJOBMSGQSZ is numeric and is initialized to 16 K bytes.

QJOBMSGQTL. Truncation level for the job message queue. QJOBMSGQTL is numeric and is initialized to 24 K bytes.

QCURSYM. Currency symbol. QCURSYM is character and can be any character except -, &, *, or 0. It is initialized to $.

COMMAND LIST

This is a list of commands related to system values. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL.*

Descriptive Name	Command Name	Function
Change System Value	CHGSYSVAL	Changes the value of a system value.
Retrieve System Value	RTVSYSVAL	Retrieves a system value and places it into a control language variable.
Display System Value	DSPSYSVAL	Displays a system value and its attributes.

Chapter 18. Security

Security controls who can use the system, who can use objects, and what rights of access each user has. Security is the prevention of access to objects by unauthorized persons. Integrity is the protection of objects from accidental destruction or alteration.

Security controls:

- Access to the system by optionally requiring the user to identify himself through the use of a password when signing on or through the use of job descriptions for running batch jobs

- Resources on the system by requiring that users be authorized to use system resources such as commands and devices

- Data in the system by requiring that users be authorized to use objects such as files and programs

This control is accomplished through security at the system level and the user level. The system level of security is controlled by the security officer of your system installation. Only the security officer has full authority (all object and special authority) across the system and is the only user authorized to control the installation's security procedures. The security officer controls the creating and maintaining of the basic unit of security, the *user profile*. The user profile is the user's authority to use the system; it represents the user to the system. If a user profile is deleted from the system, the user no longer has the authority to use the system.

The user level of security consists of authorizing users to use objects and system resources. Either the security officer or an object's owner can authorize nonowners to use objects. (An *owner* is the user who creates the object or who has been given ownership of the object.)

USER IDENTIFICATION

When a user signs on to the system he must use a password. (For a batch job the user name is in the job description.) The user is prompted to sign on:

```
Enter user password to sign on:
```

This user password tells the system who the user is and what user profile to use to initiate the job.

The user profile contains the following information:

- *User name*. The name by which the system identifies the user profile. When users are granted authority to objects, the user name is the name to which authority is granted. The security officer assigns this user name when the user profile is created. This name never changes.

- *Owned objects*. A list of all objects owned by this user profile.

- *Authorized objects*. A list of all objects privately authorized to this user and what authority the user has for the objects.

- *Special authority*. Whether the user can perform save/restore operations, save the system, or control the operation of other users' jobs.

- *Storage*. The maximum amount (K bytes) of storage the user can use for storing permanent objects owned by the user.

The following information is associated with the user profile but is not contained in it.

- *User password*. The name by which the user signs on the system and by which the system determines which user profile is to be used. The security officer assigns this user password.

- *Priority limit*. The highest scheduling priority and output priority available for the user's jobs.

- *Initial program*. The program to be invoked automatically when the user uses the IBM-supplied control language processing program.

A user profile can be created for each user of the system, or a user profile can be created for several users. In addition, you can use the user profiles shipped with your system:

- Security officer user profile (QSECOFR)

- Programmer user profile (QPGMR)

- Work station user profile (QUSER)

- System operator user profile (QSYSOPR)

- Programming service representative user profile (QPSR)

- Customer engineer user profile (QCE)

The QPGMR user profile is specified as the user profile for the IBM-supplied batch job description QBATCH. It contains the authority necessary for system and applications programmers. This profile can be modified or deleted. The initial user password is PGMR.

The QUSER user profile is specified as the user profile for the IBM-supplied interactive job description QCTL. It contains the authority necessary for the work station user. This profile can be modified or deleted. The initial user password is USER and the initial program is QCALLMENU.

The IBM-supplied system operator user profile QSYSOPR has the special authority of save/restore authority for all objects, save system authority, and job control authority. The system operator can save and restore all objects and change, cancel, display, hold, and release all jobs and spooling queues (if the queues can be controlled by the operator). This profile can be modified or deleted. The initial user password is SYSOPR.

The IBM-supplied programming service representative (PSR) user profile QPSR has the authority needed by the PSR to service the system's programming. The PSR should not be revoked from the system console and must be individually authorized to all display work stations. When a work station is authorized to all users (*ALL), the PSR is not included. You cannot delete the PSR user profile. The initial user password is PSR.

The IBM-supplied customer engineer (CE) user profile QCE has the authority needed by the CE to perform diagnostics and service the machine. The CE should not be revoked from the system console and must be individually authorized to all display work stations. When a work station is authorized to all users (*ALL), the CE is not included. You cannot delete the CE user profile, nor can you have more than one CE user profile. The initial program for QCE is the concurrent service monitor, which is used to diagnose system problems. The initial user password is CE.

You should restrict who can use the system by changing the user passwords for QPGMR, QUSER, QSYSOPR, QPSR, and QCE. However, only the security officer can make these changes. If you change the password for QCE, you should contact the customer engineer. For example, the security officer enters the following Change User Profile (CHGUSRPRF) commands:

```
CHGUSRPRF USRPRF(QUSER)   PASSWORD(BRONZE)
CHGUSRPRF USRPRF(QPGMR)   PASSWORD(SILVER)
CHGUSRPRF USRPRF(QSYSOPR)   PASSWORD(GOLD)
```

To use the QPGMR user profile, a user would sign on as SILVER.

In addition to the five IBM-supplied user profiles QPGMR, QUSER, QSYSOPR, QPSR, and QCE, your system is shipped with a security officer user profile (QSECOFR).

Security Officer User Profile (QSECOFR)

The user password for this user profile is SECOFR, and it should be changed to maintain integrity and security.

The security officer is the only user authorized to:

- Enroll users on the system by creating a user profile
- Change a user profile
- Display a list of user names and passwords

In addition, the security officer can:

- Remove a user from the system by deleting a user profile
- Grant and revoke authority for all system resources such as devices, objects, and commands
- Display the contents of other users' user profiles
- Display object authority information for all objects

You cannot delete the security officer user profile, nor can you have more than one security officer user profile. No other user profile can be granted all object authority.

Although the security officer has all object authority, he must be privately authorized to and revoked from all work stations other than the system console. When a work station is authorized to all users (*ALL), the security officer is not included. (This controls the power of the security officer.) The security officer cannot be revoked from the system console because he would be prevented from signing onto the system.

The following commands are authorized to the QSECOFR user profile only. No other user profile can use these commands.

Change User Profile (CHGUSRPRF)
Create User Profile (CRTUSRPRF)
Display Authorized Users (DSPAUTUSR)

OBJECT OWNERSHIP

All objects have owners. Initially, the user who creates the object is the owner. However, the owner, the security officer, or a user with object existence rights can transfer ownership to another user. The owner and the user with object existence rights must have add rights for the user profile to which ownership is being transferred and delete rights to the current owner's user profile. See *Object Authority* for an explanation of the types of authority.

If a user profile is being deleted because a user is no longer authorized to use the system, the objects owned by that user must be assigned to new owners. Otherwise, the objects must be deleted, because an object cannot exist on the system without an authorized owner and a user profile cannot be deleted if it owns objects.

The Change Object Owner (CHGOBJOWN) command is used to transfer ownership. In the following example the present owner is changing the owner of the order department general menu program ORD040C.

```
CHGOBJOWN   OBJ(ORD040C.DSTPRODLB) OBJTYPE(PGM)
            NEWOWN(BWALTON)
```

The object description changes like this:

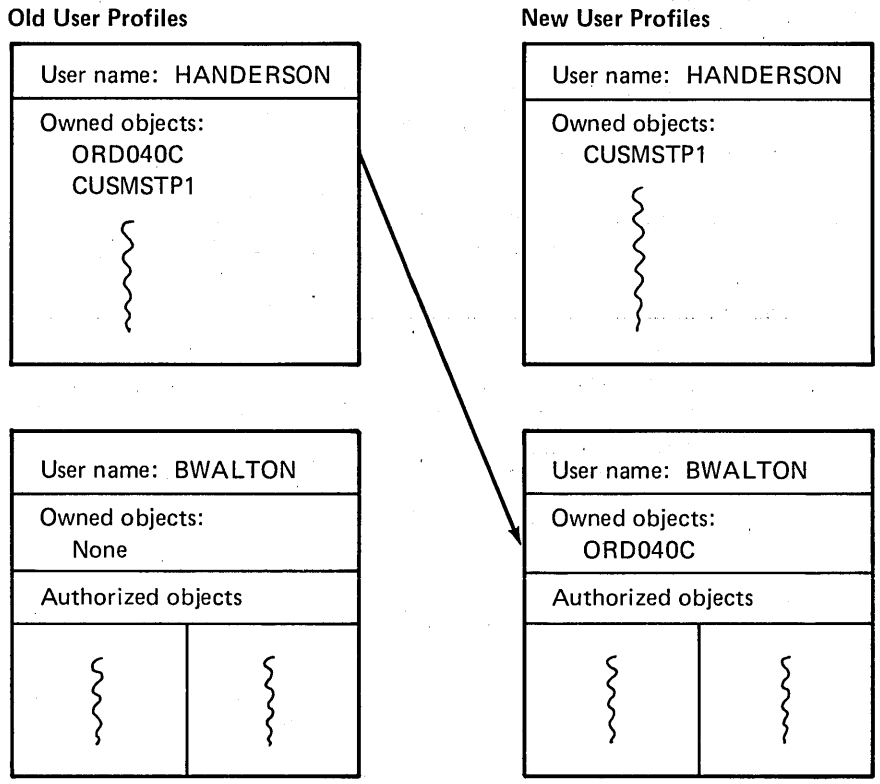
Old Object Description

Object name: ORD040C
Owner: HANDERSON
.
.
.

New Object Description

Object name: ORD040C
Owner: BWALTON
.
.
.

The user profiles change like this:



OBJECT AUTHORITY

Object use can be authorized privately or publicly. Objects that are privately authorized are available only to specific users. Object rights that are publicly authorized are available to all users.

An object's owner and the security officer have all authority for an object. They can both grant authority for an object to other users of the system and revoke authority from other users of the system.

There are two major groups of rights: object rights and data rights.

Object rights control how the user can use the entire object:

- *Object existence rights*. The right to delete, save, free the storage of, restore, and transfer ownership of an object. For example, to delete a program you must have object existence rights for the program.
- *Object management rights*. The right to move, rename, grant authority to, revoke authority from, and change the attributes of an object. For example, to move an object from one library to another you must have object management rights for the object.
- *Operational rights*. The right to use an object and look at its description. These rights vary according to object type. For example, to compile a program using an externally described data file, you must have operational rights to the file. To execute the program and read records from the file, you must also have read rights to the file.

Data rights control how the user can use the data contained in the object:

- *Read rights*. The right to read the entries in an object. For example, to read records from a file you must have read rights for the file.
- *Update rights*. The right to change the entries in an object. For example, to update a record in a file you must have update rights for the file.
- *Add rights*. The right to add an entry to an object. For example, to place a program in a library you must have add rights for the library.
- *Delete rights*. The right to delete an entry from an object. For example, to delete a record from a file you must have delete rights for the file.

Note: Each user profile is granted object management, read, add, and delete rights for itself so that the user can perform functions such as creating objects, deleting objects, transferring ownership of objects, and displaying authorized objects. Because a user has these rights, the user can grant and revoke to and from other users the read, add, and delete rights to his user profile.

GRANTING OBJECT AUTHORITY

Both object rights and data rights can be granted to users through the Grant Object Authority (GRTOBJAUT) command. To grant object and data rights, the user must be the security officer, the owner, or a user with object management rights and any right to be granted. Only the security officer and object owner can grant object management rights. In the following example you want to authorize RSMITH, BJONES, TBROWN, and WDOUGLAS to use the order department general menu. These users are to be given operational rights, which means they can execute the program that displays the menu.

```
GRTOBJAUT OBJ(ORD040C.DSTPRODLB) OBJTYPE(PGM)
          USER(RSMITH BJONES TBROWN WDOUGLAS)
          AUT(*OPER)
```

When an object is created, the authority to it can be designated as public, private, or normal. A publicly authorized object is one for which all users have all authority. A privately authorized object is one restricted to the use of the owner, security officer, and other users to whom authority was granted privately through the Grant Object Authority (GRTOBJAUT) command. Rights, other than publicly authorized rights, are kept in the user profile to which the rights are granted. Publicly authorized rights for the object are kept in the object. An object designated as normal is an object for which all users have only partial authority. Normal authority varies according to object type. The following chart shows what rights are granted when an object is designated as normal (indicated by N) and what rights (indicated by X and N) apply to each object type. Normal authority includes operational and any data rights needed to perform common functions on an object. Object existence and object management rights are never granted as part of normal authority.

	Object Rights			Data Rights			
Object Type	Object Existence	Object Management	Operational	Read	Add	Update	Delete
Class	X	X	N				
Command	X	X	N				
Control unit description	X	X	N				
Data area	X	X	N			X	
Device description	X	X	N				
Edit description	X	X	N				
File							
Device	X	X	N				
Logical	X	X	N	N	N	N	N
Physical	X	X	N	N	N	N	N
Job description	X	X	N				
Job queue	X	X	N	N	N	X	X
Library	X	X	N	N	X	N	N
Line description	X	X	N				
Message file	X	X	N	N	N	X	X
Message queue	X	X	N	N	N	X	N
Output queue	X	X	N	N	N	X	X
Program	X	X	N				
Print image	X	X	N				
Subsystem description	X	X	N				
Table	X	X	N			X	
User profile	X	X		X	X		X

The following chart shows what rights a user needs to perform certain CPF functions. The objects listed under the *Rights* columns indicate which objects require the rights.

Function	**Rights**						
	Object Existence	**Object Management**	**Opera-tional**	**Add**	**Read**	**Update**	**Delete**
Create object				USRPRF LIB	LIB		
Delete object 1	OBJ				LIB		
Move object		OBJ		To LIB	LIB		From LIB
Rename object		OBJ			LIB	LIB	
Grant authority		OBJ or be owner of OBJ			LIB		
Revoke authority		OBJ or be owner of OBJ			LIB		
Transfer ownership	OBJ			To USRPRF	LIB		Owner's USRPRF
Display object 2			OBJ		LIB OBJ if PGM		
Display object description 3					LIB		
Display object authority		OBJ			LIB		
Change object 4		OBJ			LIB		
Call to program			PGM		LIB		
Save 5	OBJ or be owner of OBJ		DEVD LIND CUD		LIB		
Free storage 5	OBJ or be owner of OBJ				LIB		
Restore 5	OBJ or be owner of OBJ		DEVD LIND CUD		LIB		

Legend:

USRPRF	User profile
LIB	Library
OBJ	Object
PGM	Program
DEVD	Device description
LIND	Line Description
CUD	Control unit description

1 To delete libraries, device descriptions, control unit descriptions, line descriptions, and user profiles, no authority is needed for a library.

2 To display device descriptions, control unit descriptions, line descriptions, and user profiles, no authority is needed for a library.

3 Any or some authority is needed for the object.

4 To change user profiles, device descriptions, control unit descriptions, and line descriptions, no authority is needed for library.

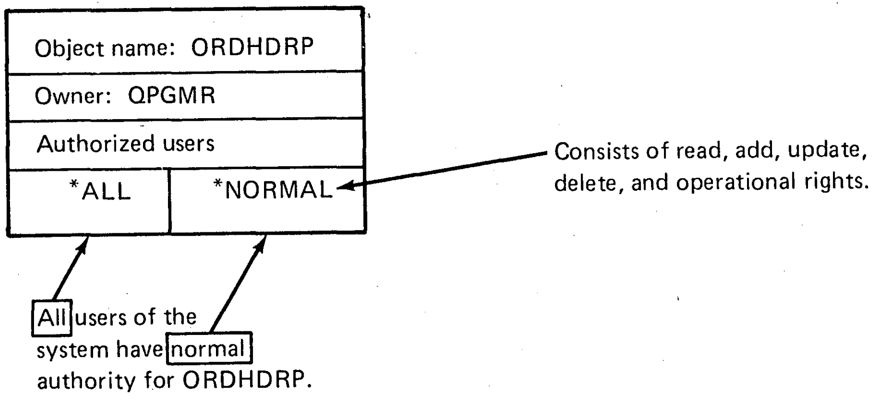
5 You must have save/restore authority (*SAVRST) or save system (*SAVSYS) authority.

Note: Because libraries are often public and objects being used are often public or owned by you, you should usually have the authority required for any reasonable function. Unusual functions executed on another user's private object requires that the user privately grant you authority.

The PUBAUT parameter on the create object commands indicates what authority the public (all users) has for an object. For example, the following command gives normal authority to all users for a physical file ORDHDRP.

```
CRTPF  FILE(ORDHDRP) SRCFILE(QDDSSRC) PUBAUT(*NORMAL)
```

The object description for ORDHDRP is:


Object name: ORDHDRP
Owner: QPGMR
Authorized users
*ALL
*NORMAL
Consists of read, add, update, delete, and operational rights.
All users of the system have normal authority for ORDHDRP.

Specific rights instead of all rights can be given to the public. In this case, public authority (*ALL) should not be specified in the PUBAUT parameter; instead, normal authority (*NORMAL) or private authority (*NONE) is specified. Then, the GRTOBJAUT command is used to grant a specific right publicly to users. The USER parameter in the GRTOBJAUT would be *PUBLIC and the AUT parameter would indicate which right is being authorized publicly.

RUNNING A PROGRAM UNDER AN OWNER'S USER PROFILE

When you create a program to be used by other users, you must authorize the user not only to the program but also to the objects (such as files) associated with the program. You can grant each user the specific rights he needs to the objects. However, by specifying, when the program is created, that the program is to always run under the owner's user profile, private authority does not have to be granted. Other users have authority for the objects only when they are executing the program and then only within the program or in subsequent programs invoked by the program.

To specify that a program is to run under an owner's user profile, you specify the following parameter and value on a create program command:

```
USRPRF(*OWNER)
```

Only the security officer can transfer ownership of a program that executes under the owner's user profile.

CREATING USER PROFILES

Only the security officer can create and change a user profile. He uses the Create User Profile (CRTUSRPRF) command to create the profile.

In the following example, the security officer creates a user profile for a work station user.

```
CRTUSRPRF USRPRF(RSMITH) PASSWORD(PRIZE)
          SPCAUT(*NONE) MAXSTG(*NOMAX)
          PTYLMT(5) INLPGM(ORD040C.DSTPRODLB)
          TEXT('Bob Smith - Order dept 348')
```

Initially, the user profile for RSMITH looks like this:

User name: RSMITH
Owned objects
None
Authorized objects
None
Initial program: ORD040C.DSTPRODLB
User description: Bob Smith – order dept 348

As RSMITH is given object authority or creates objects, the objects he is authorized to and owns and the type of rights given are recorded in his profile.

To change a user profile and its associated information, the security officer uses the Change User Profile (CHGUSRPRF) command. The password, special authority, storage, priority limit, initial program, and text can be changed. In the following example, the security officer changes the user profile of RSMITH. The initial program changes from ORD040C to QCALLMENU.

```
CHGUSRPRF USRPRF(RSMITH) INLPGM(QCALLMENU)
          TEXT('Bob Smith - Accounting dept 512')
```

However, RSMITH still must be authorized to QCALLMENU through the Grant Object Authority (GRTOBJAUT) command and can have his authority for ORD040C revoked (see *Revoking Object Authority*).

After all changes are made, including authority changes, the user profile for RSMITH looks like this:

User name: RSMITH	
Owned objects	
None	
Authorized objects	
QCALLMENU	*NORMAL
Initial program: QCALLMENU	
User description: Bob Smith – accounting dept 512	

REVOKING OBJECT AUTHORITY

Object authority can be revoked from a user by the security officer, the object's owner, or a user having object management rights. However, only the security officer or the object's owner can revoke public authority and object management rights. A user with object management rights can revoke only his rights.

In the following example you revoke the authority RSMITH has for the order department general menu.

```
RVKOBJAUT OBJ(ORD040C.DSTPRODLB) OBJTYPE(PGM)
          USER(RSMITH) AUT(*NORMAL)
```

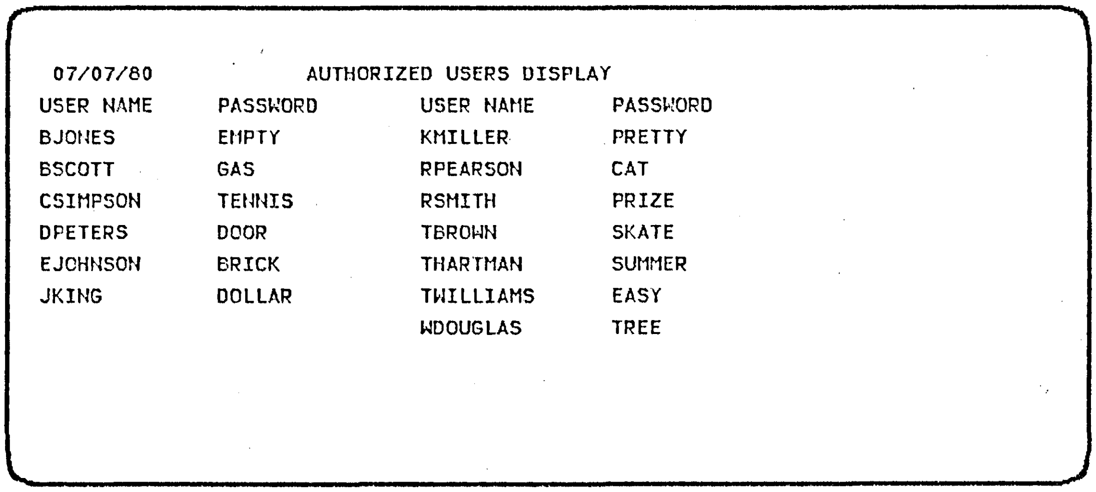
DISPLAYING SECURITY INFORMATION

Three displays contain security information:

- Authorized users display
- User profile display
- Object authority display

Authorized Users Display

The authorized users display is a list of user names and their associated passwords that only the security officer can ask for. The Display Authorized Users (DSPAUTUSR) command is used to display or print the list alphabetically by user name or user password. The following is an example of the authorized users display in which the user names are listed alphabetically.



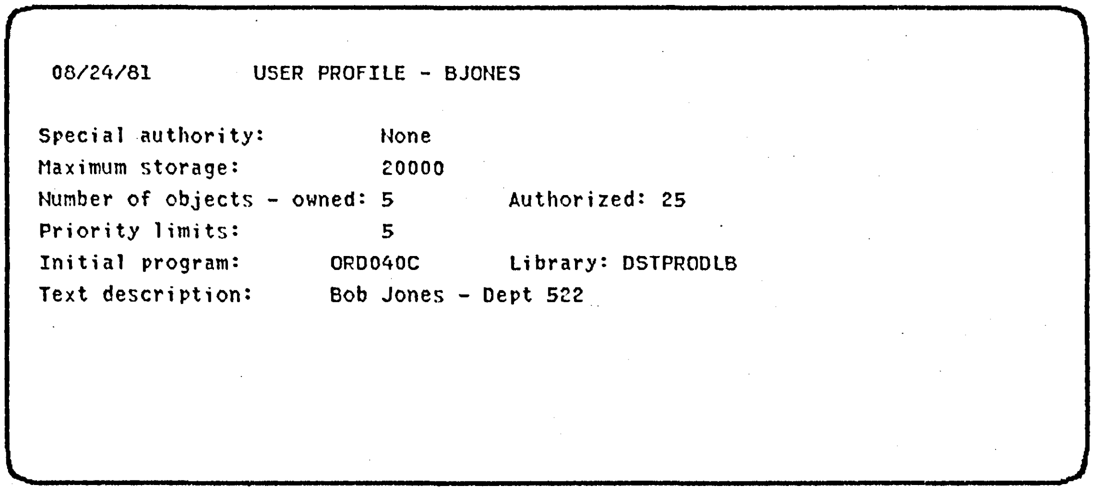
```
 07/07/80              AUTHORIZED USERS DISPLAY
USER NAME     PASSWORD         USER NAME      PASSWORD
BJONES        EMPTY            KMILLER        PRETTY
BSCOTT        GAS              RPEARSON       CAT
CSIMPSON      TENNIS           RSMITH         PRIZE
DPETERS       DOOR             TBROWN         SKATE
EJOHNSON      BRICK            THARTMAN       SUMMER
JKING         DOLLAR           TWILLIAMS      EASY
                               WDOUGLAS       TREE
```

User Profile Display

You can display the entire contents of a user profile or only one of the following parts of a user profile:

- Basic part: User name, special authority authorized to user, storage, priority limit, initial program name, text description, number of objects owned by the user, and number of objects authorized to the user
- Commands to which the user is privately authorized
- Devices to which the user is privately authorized
- Objects to which the user is privately authorized and what his authority for each object is
- Objects owned by the user

Any user who is authorized to the Display User Profile (DSPUSRPRF) command and has read rights for the user profile can ask for the user profile display. The information can also be printed. The following is an example of the user profile display in which a basic display for BJONES is displayed.

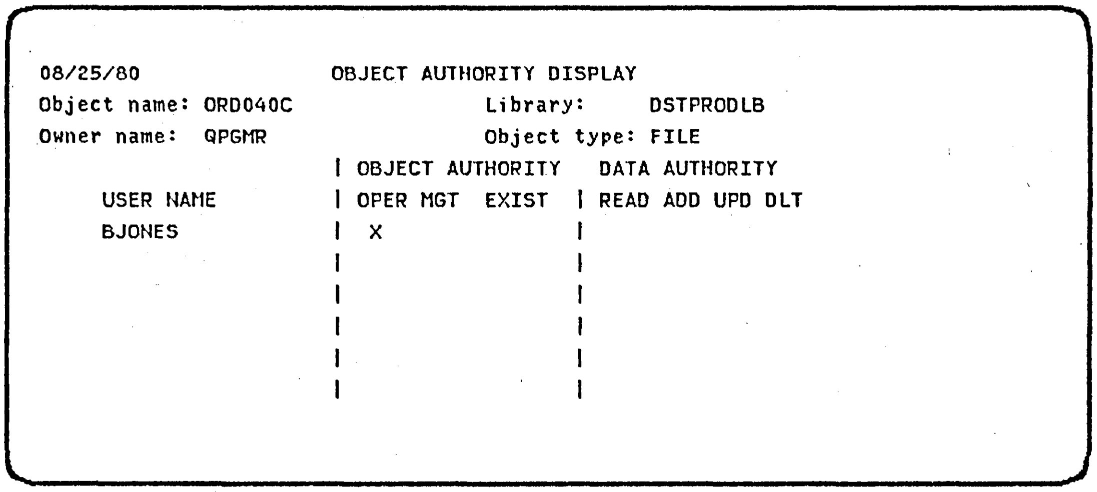


```
 08/24/81        USER PROFILE - BJONES

Special authority:         None
Maximum storage:           20000
Number of objects - owned: 5          Authorized: 25
Priority limits:           5
Initial program:       ORD040C        Library: DSTPRODLB
Text description:      Bob Jones - Dept 522
```

Object Authority Display

The security officer, the object's owner, or a user with object management rights can ask for a display of: (1) the rights that have been granted for a specific object, and (2) to whom those rights were given. This display can be printed.

The following is an example of the object authority display for the object ORD040C in the library DSTPRODLB.



```
08/25/80                 OBJECT AUTHORITY DISPLAY
Object name: ORD040C                 Library:    DSTPRODLB
Owner name:  QPGMR                   Object type: FILE
                         | OBJECT AUTHORITY   DATA AUTHORITY
     USER NAME           | OPER MGT  EXIST  | READ ADD UPD DLT
     BJONES              |  X               |
                         |                  |
                         |                  |
                         |                  |
                         |                  |
                         |                  |
```

COMMAND LIST

This is a list of commands related to security. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual–CL*.

Descriptive Name	**Command Name**	**Function**
Change Object Owner	CHGOBJOWN	Transfers ownership of an object to a new owner.
Display Authorized Users	DSPAUTUSR	Displays a list of user names and their user passwords.

User Profile

Descriptive Name	Command Name	Function
Create User Profile	CRTUSRPRF	Identifies a user to the system by creating his user profile.
Delete User Profile	DLTUSRPRF	Deletes a user from the system by deleting his user profile.
Change User Profile	CHGUSRPRF	Changes the attributes of a user profile.
Display User Profile	DSPUSRPR	Displays the contents of a user profile.

Object Authority

Descriptive Name	Command Name	Function
Grant Object Authority	GRTOBJAUT	Grants authority for objects to users.
Revoke Object Authority	RVKOBJAUT	Revokes authority for objects from users.
Display Object Authority	DSPOBJAUT	Displays the list of users authorized to use an object and the rights to that object.

Chapter 19. Save/Restore

Save/restore lets you save backup copies of all objects on your system so that you can recover from program or system failure. Save/restore can also be used to keep infrequently used objects offline so that online storage is available for more frequently used objects.

You can save and restore:

- A single object in a library
- A group of objects by generic name
- A group of objects by generic name and object type:
  - Command definitions
  - Message files
  - Tables
  - Programs
  - Print images
  - Classes
  - Edit descriptions
  - Subsystem descriptions
  - Files
  - Job descriptions
  - Data areas
- An entire library
- The system (QSYS, device configuration, and user profiles)

When an object is saved the following information is saved:

- Label
  - Object name
  - Object type
  - Date and time of the save
  - Storage required in the system
  - Text description
- Owner name
- Any public authority
- The object itself

You can save and restore objects while normal system operation continues. However, if an object is being used for updating or is allocated exclusively to a job, it cannot be saved or restored while being used. In addition, an object cannot have its storage freed if it is being used in a job.

If you are saving the entire system, the system must be dedicated to the save. Other functions cannot be performed at the same time.

When an object is saved, it is not removed from the system. The object still exists on the system. That is, it still occupies storage on the system and can be used normally. Only a copy of it is saved.

If you do not want to keep a data base file or a program on the system, you can make the storage it occupies available for other use. This is called freeing storage. You free the storage when you save the object by using the STG(*FREE) parameter.

Notes:

1. If you free storage, the object is offline and must be restored to be used.
2. Freeing storage is different from deleting an object in that when you delete an object all information about the object is removed from the system. To use the object again you must create it again and, if the object is private, grant authority to users.

You can restore objects to a system on which the object still exists and on which the object does not exist online. The object might have had its storage freed or been deleted or might have never existed on the system you are restoring it to. By restoring objects from one system onto another system, you can move objects from one system to another.

When an object is restored for which its storage has not been freed, the object existing on the system is replaced by the saved object. If the object does not exist on the system, the system allocates storage for it and restores it. If only selected objects from a library were saved, only those objects, not the entire library, can be restored from a library that has been saved.

In addition, you can restore objects to a library other than the library from which they were saved.

The save/restore history information contains the following about each object:

- Date and time of the last save
- Date and time of the last restore
- Volume identifiers of volumes containing the most current saved version
- The number of intermediate magazines used in the last save (the number of magazines used between the first and last magazines)

Restoring an Application

To move an application from one system to another, use the restore function. The following shows the order in which you must restore the application.

1. Enroll users on the system.

2. Restore the libraries containing the application. Either the system operator or a user who owns all the objects involved can restore the libraries. If the system operator restores the objects, the security officer becomes the owner of any object whose owner does not exist on the system.

   You may have to change the library names if the names are also names of libraries on the new system. Or, you can delete the libraries that already exist on the system. If you change the library names, the programs must be changed to reflect the new library names.

3. Authorize users to the necessary objects.

4. Use override commands to change the programs to reflect changes due to the new system.

Security Considerations

To save, free the storage of, and restore all objects on the system, you must have save system authority. To save, free the storage of, and restore objects to which your user profile has object existence rights, you must have save/restore authority. Both save system and save/restore authority are special authority that can be granted by the security officer when a user profile is created or changed.

When an object is restored to the system but its owner no longer exists on the system, the security officer becomes the owner of the object.

If a private object is being restored and the object's storage was freed, the owner does not have to grant authority again to the users to use the object. If the object was saved as a public object, it is restored as a public object. If an object is restored that had been deleted or never existed on the system, all private authority must be granted again.

Diskette Considerations

Diskettes are the save/restore medium. Objects are saved on diskettes in the diskette magazine drive. They are then taken out of the drive and stored offline.

The diskette magazine drive can hold two magazines, each containing ten diskettes, and three slots for manually inserting diskettes. The magazines are called *M1 (first magazine) and *M2 (second magazine). The diskettes in a magazine are numbered from one through ten. The slots are called *S1 (first slot), *S2 (second slot), and *S3 (third slot).

Before you can use a diskette on the system, it must be initialized and have volume labels written on it. The Initialize Diskette (INZDKT) command does this for you.

To use the diskette magazines for saving and restoring, the following conventions are followed:

- The volume identifier consists of two through six characters of which the first one through five characters are the magazine identifier and must be the same for all diskettes in a magazine. The last character identifies the position of the diskette in the magazine and must be one through ten, where ten is designated as 0.
- A magazine is assumed to be full (contains 10 diskettes).
- Before a diskette magazine is used in a save operation, all of the diskettes in the magazine, beginning with the first diskette on which the saved copy is to be written, must be clear. (Use the Clear Diskette (CLRDKT) command.)

SAVING A LIBRARY

You can save all libraries or a single library.

The following example shows how to save a single library. In this example, the general purpose library QGPL is saved.

```
SAVLIB LIB(QGPL) LOC(*M12) VOL(*MOUNTED) STG(*KEEP)
```

The library is saved on magazines 1 and 2 on the volumes mounted. The storage for QGPL is not freed; therefore, QGPL still exists on the system.

If you had wanted to save all libraries with the exception of the system library QSYS, you would have specified *ALL for the library name.

RESTORING A LIBRARY

You can restore a library or all libraries (except QSYS) to the system. Objects in the library can be either old or new. Old objects are those that were saved on your system and have not been deleted. New objects are those that were saved on another system or were deleted from your system after they were saved. You have four options concerning new and old objects when you restore a library.

1. Only old objects are replaced in a library.
2. Only new objects are added to a library. The old objects are not replaced.
3. Old objects are replaced and new objects are added to a library.
4. Only those objects whose storage has been freed are restored.

If you specify a volume identifier, you can specify a save date and save time. This lets you specify exactly which saved version you want. If you do not specify a save date or save time, the first version encountered on the volume is restored.

When you restore a library, you do not have to specify the specific volume on which the library was saved. You can specify that the most current saved version be used for the restore.

In the following example, you restore the general purpose library QGPL.

```
RSTLIB SAVLIB(QGPL) LOC(*M12)
    VOL(*CURRENT) OPTION(*MIXED) RSTLIB(*SAVLIB)
```

The saved library is on magazines 1 and 2. The most current version is restored. If the most current version was not mounted, a message is issued to the system operator to mount the correct volume (diskette). Old objects are replaced, and new objects are added to the library (*MIXED).

SAVING AND RESTORING A GROUP OF OBJECTS

You can save a group of objects by a generic name or by generic name and type. However, all the objects must be from the same library.

In the following example, you save some of your order entry files that are in library DSTPRODLB.

```
SAVOBJ OBJ(ORD*) LIB(DSTPRODLB) OBJTYPE(FILE)
    LOC(*M12) VOL(*MOUNTED) STG(*KEEP)
```

The files are saved on magazines 1 and 2 on the volumes mounted. The storage for these files is not freed. The objects still exist in DSTPRODLB.

When you restore this group of objects you use the generic name ORD* again. If you do not have a record of the location of a saved object, you can get this information by displaying the object description. You must use the Display Object Description (DSPOBJD) command and ask for a full description (DETAIL(*FULL)). See Chapter 3, *Objects* for a description of the command and the resulting display.

SAVING A SINGLE OBJECT

You can save a single object or more than one object (not a generic group).

In the following example, you save a single object, your field reference file.

```
SAVOBJ OBJ(DSTREF) LIB(DSTPRODLB) OBJTYPE(FILE)
    LOC(*S1) VOL(SVVOL1) STG(*KEEP)
```

The file is in DSTPRODLB and remains there because its storage is not freed. It is saved on volume SVVOL1, which was placed in slot 1 (*S1). If SVVOL1 is not in slot 1 when the save request is entered, the system operator receives a message to mount the volume in the slot.

SAVING AND RESTORING AN ENTIRE SYSTEM

To save an entire system you must save all the user libraries (including QGPL) and the system library (QSYS).

Use the Save Library (SAVLIB) command to save all user libraries and the general purpose library QGPL (LIB(*ALL)).

Use the Save System (SAVSYS) command to save the system library QSYS, the system's device configuration, and all user profiles.

```
SAVSYS LOC(*M12) VOL(*MOUNTED)
```

The system is saved on the volumes mounted in magazines 1 and 2. When the diskettes are full, the system operator receives a message to take the full diskettes out and to mount more diskettes.

Note: The magazine on which the system is saved must be cleared.

To restore an entire system you must do the following (in the same order):

1. Install the CPF

2. Restore the user profiles

3. Restore all the user libraries and QGPL

4. Restore object authority

After the CPF is installed, the user profiles must be restored before the libraries can be restored. An object or library cannot be restored unless an owner exists on the system for the object or library. Besides restoring the user profiles, this command loads the object authority information needed to restore object authority. The object authority part of the user profiles is not restored when the profiles are restored.

To restore the user profiles, use the Restore User Profiles (RSTUSRPRF) command and specify the location and volume.

```
RSTUSRPRF LOC(*M12) VOL(*MOUNTED)
```

After the user libraries are restored, the user authority for objects must be restored to the user profiles. User authority could not be restored previously because you cannot give authority for an object that does not exist on the system.

To restore authority, use the Restore Authority (RSTAUT) command. There are no parameters on this command.

COMMAND LIST

This is a list of commands related to save/restore. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL*.

Descriptive Name	Command Name	Function
Save System	SAVSYS	Saves the IBM-supplied libraries (except QGPL), the the device configuration, and all user profiles on the save/restore medium.
Restore Authority	RSTAUT	Restores user profile authority to objects.
Restore User Profile	RSTUSRPRF	Restores user profiles to the system.

Object

Descriptive Name	Command Name	Function
Save Object	SAVOBJ	Saves objects on the save/restore medium.
Restore Object	RSTOBJ	Restores objects to the system.

Library

Descriptive Name	Command Name	Function
Save Library	SAVLIB	Saves a single library or all libraries on the save/restore medium.
Restore Library	RSTLIB	Restores a single library or all libraries to the system.

Chapter 20. Service

In most cases, service personnel can service your system while the system performs normal data processing activities. Most service functions are primarily designed for service personnel, but some functions are available to you and are described in this chapter.

If you are authorized to the system operator user profile, you can perform the following service functions to help determine the nature of the problem and whose problem it is (IBM's or yours):

- Copy the error log
- Trace a job and the internal machine processing
- Start a confidence check of devices
- Verify a printer
- Start the problem determination procedures

For information about applying program changes, see the *Program Product Installation Guide.*

COPYING THE ERROR LOG

Error log data consists of machine checks, device errors, and volume statistical data. You can print a copy of this data for service personnel so they can analyze it without tying up your system. You can continue using your system while the copy is executing.

The error log can be copied by type of error log data:

- All error log data
- A summary of error log data
- Only machine check data
- Only device error data
- Only volume statistical data

If only device error data is copied, you can specify that the data for all devices or only for a particular device is to be copied.

If only volume statistical data is copied, you can specify that the data is to be deleted from the error log after it is copied.

In addition, you can select error log data (except volume statistical data) to be copied by specifying a start time and end time. The start time is the time and day the first piece of data to be copied was placed in the error log. The end time is the time and day the last piece of data to be copied was placed in the error log.

Use the Copy Error Log (CPYERRLOG) command to copy the data. The following CPYERRLOG command copies all machine check data placed in the error log on 10 January 1981. Because all data for that day is copied, *AVAIL was specified for time.

```
CPYERRLOG TYPE(*MCH) PERIOD((*AVAIL 01:10:81) (*AVAIL 01:11:81))
```

TRACING A JOB AND THE INTERNAL MACHINE PROCESSING

Traces collect information necessary to analyze a problem. You can trace a job or trace internal machine activities.

Tracing a job involves recording the calls and returns of programs. The trace records contain:

- Operation
  - Call
  - Return
  - Transfer control
  - Event handler
  - External exception handler
  - Invocation destroyed

- Program name of the program on which the operation was performed (such as the program called or returned to)

- Name of the library containing the program

- Invocation number

- Instruction number of the program receiving control

- Instruction number of the program giving control

- Time stamp

- Relative record number

When a trace record is generated, it can be sent to a user exit program for handling (examining and altering). You must create this program if you want it. By using such a program you can suppress a trace record by filling it with blanks. The trace record is passed to the program in the form of a single character string. The exit program can contain the dump commands (see Chapter 15, *Testing*); output from dump commands is spooled to the job requesting the trace. Output from other commands that create spooled output is spooled to the job executing the command.

When you start tracing a job you can set up storage to contain the trace records and specify what is to happen when all the storage has been used. Two things can happen:

- The trace stops
- Old trace records are replaced with new records (this is called wrapping)

The Trace Job (TRCJOB) command starts a trace. You can also use it to stop a trace. The following TRCJOB command stops a trace.

```
TRCJOB SET(*OFF)
```

To trace the internal activities of the machine, use the Trace Internal (TRCINT) command. However, service personnel should tell you what to specify in the command.

STARTING A CONFIDENCE CHECK

You can use a confidence check to ensure that the I/O devices, the machine, and CPF are functioning together. A confidence check does the following:

- For MFCU input, cards are read; for MFCU output, cards are punched and, optionally, printed.
- For a 5211 Printer, pages are printed.
- For diskettes, diskettes are written on and then read. A diskette must be in manual slot one. This diskette must be initialized and have a volume label.

You can put a time limit on how long the devices are to run. Also, you can specify which devices are to run. You use the Start Confidence Check (STRCNFCHK) command to start the check and specify the devices and time.

The following STRCNFCHK command starts the printer PRINTER and runs it for 1200 seconds.

```
STRCNFCHK DEV(PRINTER) TIME(1200)
```

VERIFYING A 5256 PRINTER

To determine if a 5256 Printer is functioning properly, you run a test pattern. This test pattern is printed as many times as you want.

Use the Verify Printer (VFYPRT) command. All you need specify is the printer name and, optionally, how many times you want the test pattern printed. By default, the test pattern is printed once. The following VFYPRT command verifies the 5256 Printer PRINTER2 by printing the test pattern twice.

```
VFYPRT DEV(PRINTER2) TIMES(2)
```

STARTING THE PROBLEM DETERMINATION PROCEDURES

To determine the nature of a problem and whose problem it is (yours or IBM's), use the IBM-supplied problem determination procedures. The Start Problem Determination Procedures (STRPDP) command starts the initial procedures.

COMMAND LIST

This is a list of commands related to service. It is presented here to help you select the appropriate command for the function you want and to help you determine which command you might need to reference in the *CPF Reference Manual—CL*.

Descriptive Name	**Command Name**	**Function**
Copy Error Log	CPYERRLOG	Prints a copy of the error log data.
Start Confidence Check	STRCNFCHK	Starts a confidence check.
Start Problem Determination Procedures	STRPDP	Starts the problem determination procedures.
Trace Internal	TRCINT	Traces the internal activities of the machine.
Trace Job	TRCJOB	Traces a job.
Verify Printer	VFYPRT	Verifies that a printer is functioning properly by running a test pattern.

List of Abbreviations

ASCII	American National Standard Code for Information Interchange
CE	Customer engineer
CL	Control language
CPF	Control Program Facility
CPP	Command processing program
DDS	Data description specifications
EBCDIC	Extended binary coded decimal interchange code
FIFO	First-in-first-out
HLL	High-level language
I/O	Input/output
LIFO	Last-in-first-out
ODP	Open data path
PSR	Programming service representative
SEU	Source Entry Utility

Glossary

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

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

add rights: The authority to add an entry to an object.

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

attribute character: A character associated with a field in a display device file that defines how the field is displayed (such as underlined, blinking, or high intensity). This character is displayed as a blank and usually precedes the first displayed character of the field.

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

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

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 space other than main storage. Auxiliary storage is located on 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.

breakpoint: A place in a program (specified by a command or a condition) where the system halts execution so that the user can display variables, modify variables, or modify the execution sequence of the program. The user can use breakpoints to test his programs.

CL variable: A program variable that is declared in a control language program and is available only to that program.

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

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

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

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

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

control unit description: An object that describes, to the system, the features of a control unit that is either directly attached to the system or attached to a communications line.

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

data area: An object used to communicate data such as CL variable values between the programs within a job and between jobs.

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

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

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

data transformation: Changing the form of data according to specific rules as data is moved between the data base and the using program. Includes changing the data type and length, and protecting only certain fields.

data type: An attribute used for defining data as either numeric or character.

delete rights: The authority to delete an entry from an object.

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

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

edit code: A letter or number indicating that editing should be done according to a predefined pattern. Editing can include zero suppression and punctuation.

edit description: An object that describes a user-defined edit code.

edit word: A word indicating how editing should be done.

entry job: A job that is the result of a user transferring from one subsystem to another.

external storage: Data storage other than main or auxiliary storage.

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

field: A portion of a data record that serves as the basic unit of data transfer to and from files.

field reference file: A physical file whose record format describes the fields used by a group of files but which contains no members. The field descriptions in the field reference file can be referred to when data description specifications for other files are written.

file: A set of related records treated as a unit and including descriptive information about the records.

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

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

first-level message: The initial message presented to the user containing general information or designating an error.

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

high-level language: A programming language that relieves the programmer from the rigors of machine level or assembler level programming. RPG III is an example of a high-level language. Abbreviated HLL.

history log: A log of information about system status and events.

horizontally displayed records: Records that are grouped in a display so that more than one record of the same record format is displayed on each display line.

impromptu message: A message created when it is sent. Contrast with *predefined message*.

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: Pertaining to a program or system that alternately accepts input and then responds. An interactive system is conversational, that is, a continuous dialog exists between the user and the system.

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

internal storage: All main and auxiliary storage in the system.

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

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

job log: A record of requests submitted to the system by a job and the messages related to them. The job log is maintained by the CPF.

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

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

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

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

line description: An object that describes a communications line to the system.

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.

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.

menu: A type of work station display in which a list of options is shown to the user. From this menu the user selects the options he wishes.

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

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

message file: A file that contains message descriptions.

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

modified data tag: An indicator, associated with each input or output/input field in a displayed record, that is set on when data is keyed into the field. The modified data tag is maintained by the display device and can be used by the program using the file.

modulus 10, 11: A technique for validity checking that involves the association of digits with data. It is used in entering or updating fields in a data record.

multivolume file: A file that is contained on more than one diskette.

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

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

object existence rights: The authority to delete, save, free the storage of, restore, and transfer ownership of an object.

object management rights: The authority to move, rename, grant authority to, revoke authority from, and change the attributes of an object.

object owner: A user who creates an object or to whom the ownership of an object has been transferred. The object owner has complete control over the object.

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.

object user: A user who has been authorized by the object owner to perform certain functions on an object.

operational rights: Any combination of data rights authorized to a user.

option indicator: A one-character field passed from a program to the CPF with an output data record that is used to control the output function, such as controlling which fields in the record are displayed.

output/input field: A field in a display device file that is used for both output and input operations.

output queue: A list of output files that are ready to be written to an output device by a writer.

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.

predefined message: A message whose description is created independently of when it is sent and is stored in a message file. Contrast with *impromptu message*.

print image: An object that describes, to the system, the print belt on a printer.

priority: The relative significance of one program to other programs. Priority specifies the relative order of resource allocation when competition for a resource is experienced.

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

program object: One of two MI object classifications. It includes those objects used in programs that get their definition from ODT entries. (Contrast with *system object.*)

prompt: A request for information or user action. The work station user must respond to proceed.

public authority: The authority to an object granted to all users unless overridden by specific user authority.

read rights: The authority to read the entries in an object.

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

record: An ordered set of fields that make up a single occurrence of a record format.

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.

response indicator: A one-character field passed from the CPF to a program with an input record to provide information about the data record or actions taken by the work station user.

routing data: A character string that the Control Program Facility 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 job that executes in the subsystem.

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

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

security officer: The individual at an installation who can define and control the installation's security procedures.

security officer profile: The user profile that allows the security officer to control the security of an installation.

second-level message: Provides additional information to that already provided in a first-level message. Second-level messages are obtained by pressing the Help key while a first-level message is displayed.

service log: A log of information about the application of program changes and program patches, and the symptom strings resulting from errors.

shared access path: An access path used by more than one file to provide access to data common to the files. The access path specifications are contained in the description of each file that uses the access path.

shared record format: A record format that is used in more than one externally described data file.

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

space pointer: Contains addressability to an MI program object.

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

spooling: The reading and writing of input and output streams on an intermediate storage device, concurrently with job execution, and in a format convenient for later processing or output operations.

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

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

subfile: A group of records of the same record format that can be displayed concurrently at a work station. The system sends the entire group of records to the work station in a single operation and receives the group in another operation.

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

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

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

system console: That part of a computer used for communication between the operator or maintenance personnel and the computer.

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

system object: One of two MI object classifications. It includes those MI objects whose formats are not visible above MI and all objects that are defined during execution time from attribute template operands on Create instructions. These objects are referred to through system pointers. (Contrast with *program object.*)

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

system pointer: Contains addressability to an MI system object.

system value: A value that contains control information for the operation of certain parts of the system. A user can change these values to tailor the system to his working environment. System date and library list are examples of system values.

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

trace: The process of recording (1) the sequence in which the statements in a program are executed and (2) optionally, the values of the program variables used in the statements.

update rights: The authority to change the entries in an object.

user name: The name by which a particular user is known to the system.

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

validity checker: A program that tests commands for errors in the parameter values. This validity checking is done in addition to the checking done by the command analyzer.

variable: A named modifiable value. The value can be accessed or modified by referring to the name of the variable.

vertically displayed records: Records that are grouped in a display so that more than one record of the same record format is displayed concurrently, with each record beginning in the first position of a line and occupying one or more adjoining lines.

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, or both, as needed to perform his job.

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.

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

Index

READER'S COMMENT FORM

IBM System/38
Control Program Facility
Programmer's Guide

SC21-7730-0

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

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

Page Number *Error*

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

Page Number *Comment*

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.

Name ________________________________

Address ______________________________

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

SC21-7730-0

Cut Along Line

IBM System/38 Control Program Facility Prc .mer's Guide (File No. S38-36) Printed in U.S.A. SC21-7730-0

Fold and Tape | Please Do Not Staple | Fold and Tape

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

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

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

Fold and Tape | Please Do Not Staple | Fold and Tape

IBM®

International Business Machines Corporation

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

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

IBM®

International Business Machines Corporation

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

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

IBM System/38 Control Program Facility Programmer's Guide (File No. S38-36) Printed in U.S.A. SC21-7730-0

SC21-7730-0