AS/400e

Data Management

Version 4

RBAL-3000-00

AS/400e

Data Management

Version 4

RBAL-3000-00

Contents

About Data Management

This book describes the data management portion of the Operating System/400 licensed program. Data management provides applications with access to input and output file data that is external to the application. There are several types of these input and output files, and each file type has its own characteristics. In addition, all of the file types share a common set of characteristics. This book describes the characteristics and programming use of database files and spooled files.

Who should read this book

This book is intended primarily for the application programmer. This book should also be useful for those responsible for tailoring their system to use double-byte data with the data management file support.

Before using this book, you should be familiar with general programming concepts and terminology, and have a general understanding of the AS/400 system and OS/400 operating system.

AS/400 Operations Navigator

AS/400 Operations Navigator is a powerful graphical interface for Windows clients. With AS/400 Operations Navigator, you can manage and administer your AS/400 systems from your Windows desktop.

You can use Operations Navigator to manage communications, printing, database, security, and other system operations. Operations Navigator includes Management Central for managing multiple AS/400 systems centrally.

Figure 1 shows an example of the Operations Navigator display:

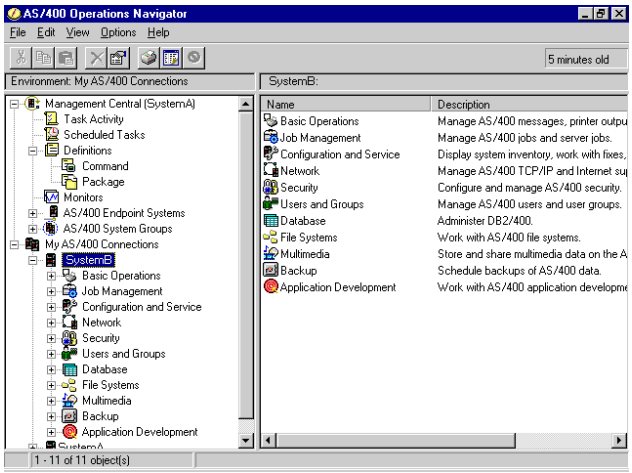


Figure 1. AS/400 Operations Navigator Display

This new interface has been designed to make you more productive and is the only user interface to new, advanced features of OS/400. Therefore, IBM recommends that you use AS/400 Operations Navigator, which has online help to guide you. While this interface is being developed, you may still need to use a traditional emulator such as PC5250 to do some of your tasks.

Installing Operations Navigator

To use AS/400 Operations Navigator, you must have Client Access installed on your Windows PC. For help in connecting your Windows PC to your AS/400 system, consult *Client Access Express for Windows - Setup*, SC41-5507-00.

AS/400 Operations Navigator is a separately installable component of Client Access that contains many subcomponents. If you are installing for the first time and you use the **Typical** installation option, the following options are installed by default:

- Operations Navigator base support
- Basic operations (messages, printer output, and printers)

To select the subcomponents that you want to install, select the **Custom** installation option. (After Operations Navigator has been installed, you can add subcomponents by using Client Access Selective Setup.)

1. Display the list of currently installed subcomponents in the **Component Selection** window of **Custom** installation or Selective Setup.
2. Select AS/400 Operations Navigator.
3. Select any additional subcomponents that you want to install and continue with **Custom** installation or Selective Setup.

After you install Client Access, double-click the **AS400 Operations Navigator** icon on your desktop to access Operations Navigator and create an AS/400 connection.

Prerequisite and related information

Use the AS/400 Information Center as your starting point for looking up AS/400 technical information. You can access the Information Center from the AS/400e Information Center CD-ROM (English version: *SK3T-2027*) or from one of these Web sites:

`http://www.as400.ibm.com/infocenter`
`http://publib.boulder.ibm.com/pubs/html/as400/infocenter.htm`

The AS/400 Information Center contains important topics such as logical partitioning, clustering, Java, TCP/IP, Web serving, and secured networks. It also contains Internet links to Web sites such as the AS/400 Online Library and the AS/400 Technical Studio. Included in the Information Center is a link that describes at a high level the differences in information between the Information Center and the Online Library.

For a list of related publications, see the “Bibliography” on page 223.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other AS/400 documentation, fill out the readers’ comment form at the back of this book.

- If you prefer to send comments by mail, use the readers’ comment form with the address that is printed on the back. If you are mailing a readers’ comment form from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid mailing.
- If you prefer to send comments by FAX, use either of the following numbers:

  - United States and Canada: 1-800-937-3430
  - Other countries: 1-507-253-5192
- If you prefer to send comments electronically, use one of these e-mail addresses:
  - Comments on books:

    RCHCLERK@us.ibm.com

    IBMMAIL, to IBMMAIL(USIB56RZ)
  - Comments on the AS/400 Information Center:

    RCHINFOC@us.ibm.com

Be sure to include the following:

- The name of the book.
- The publication number of the book.
- The page number or topic to which your comment applies.

Chapter 1. Introduction

Data management is the part of the operating system that controls the storing and accessing of data by an application program. The data may be on internal storage (for example, database), on external media (diskette, tape, printer), or on another system. Data management, then, provides the functions that an application uses in creating and accessing data on the system and ensures the integrity of the data according to the definitions of the application.

Data management provides functions that allow you to manage files (create, change, override, or delete) using CL commands, and create and access data through a set of operations (for example, read, write, open, or close). Data management also provides you with the capability to access external devices and control the use of their attributes for creating and accessing data.

If you want to make more efficient use of printers and diskette devices, data management provides the capability of spooling data for input or output. For example, data being written to a printer can be held on an output queue until the printer is available for printing.

On the IBM AS/400 system, each file (also called a file object) has a description that describes the file characteristics and how the data associated with the file is organized into records, and, in many cases, the fields in the records. Whenever a file is processed, the operating system (the Operating System/400 or OS/400 program) uses this description.

You can create and access data on the system by using these file objects. Data management defines and controls several different types of files. Each file type has associated CL commands to create and change the file, and you can also create and access data through the operations provided by data management.

File Types

The data management functions support the following types of files:

- **Database files** are files (including distributed files) whose associated data is stored permanently in the system.
- **Device files** are files that provide access to externally attached devices such as displays, printers, tapes, diskettes, and other systems that are attached by a communications line. The device files supported are:
  - Display files, which provide access to display devices
  - Printer files, which describe the format of printed output
  - Tape files, which allow access to data files on tape devices
  - Diskette files, which provide access to data files on diskette devices
  - **Intersystem communications function** (OS/400-ICF) files, hereafter referred to as ICF files, which allow a program on one system to communicate with a program on the same system or another system
- **Save files** are files that are used to store saved data on disk (without requiring diskettes or tapes).
- **Distributed data management (DDM) files** are files that allow access to data files stored on remote systems.

Each file type has its own set of unique characteristics that determines how the file can be used and what capabilities it can provide. The concept of a file, however, is the same regardless of what type of file it is. When a file is used by a program, it is referred to by name, which identifies both the file description and, for some file types, the data itself. This information is designed to help you understand the common characteristics of all file types so you can use the files to their full capabilities.

Related tasks:

See the following links for information on the tasks you can perform against files:

- Copy files
- Open files
- Secure files
- Share files
- Temporarily override the properties of a file

Chapter 2. File Processing

This chapter discusses basic aspects of processing files. Topics include:

- File operations supported by the system for use in high-level language programs
- File security considerations
- Sharing files in the same job
- Allocating file resources
- Temporarily changing a file when a program uses it
- Feedback areas maintained by the system
- Handling file errors when programs run

Data Management Operations Overview

Data management supports many operations that high-level language programs can use to process data. These include the following, which are grouped by category:

- File Preparation

  OPEN Attaches a file to a program and prepares it for I/O operations. A file may be opened for any combination of read, write, update, or delete operations.

  ACQUIRE
  Attaches a device or establishes a communications session for an open file in preparation for I/O operations.

- Input/Output

  READ Transfers a record from the file to the program. The data is made available to the program once the read has been successfully completed.

  WRITE
  Transfers a record from the program to the file.

  WRITE-READ
  Combines the WRITE and READ operations as one operation.

  UPDATE
  Updates a record with changed data. The record must have been successfully read prior to the update operation.

  DELETE
  Deletes a record in a file. The record must have been successfully read prior to the delete operation.

- Commitment Control

  COMMIT
  Guarantees a group of changes are made as a complete transaction across multiple records or multiple files.

  ROLLBACK
  Rolls back a group of changes to the point of the last commit operation.

- Completion

  FEOD Positions the file at the last volume or at the end of data. For those programs processing files for output, the last buffer of data is written. For those programs processing files for input, an end-of-file condition is forced for the next input operation.

RELEASE
Detaches a device or a communications session from an open file. I/O operations can no longer be performed for this device or session.

CLOSE
Detaches a file from a program, ending I/O operations. Any remaining data in the output buffer that has not been written will be written prior to the completion of the close.

The operations listed above have certain restrictions based on file type and language support. For example, a program may not write to a file that has been opened for read only. Similarly, a read-by-key may not be issued for an ICF file. Since file overrides can occur during processing, an operation may not be allowed for the type of file that is ultimately being processed. See Chapter 3. Using Overrides, for additional information.

Table 1 on page 4 lists the file types and the main operations that are allowed. There are additional functions supported for some file types that are accomplished by additional operations or changes to these operations. For information on these additional functions and how the operations given here apply to display, tape, and diskette files, refer to either the *Application Display Programming*, SC41-5715-00 book or the *Tape and Diskette Device Programming*, SC41-5716-01 book. For equivalent information for database, ICF, DDM, printer, and save files, refer to the 5701 book, the 5442 book, the 5307 book, the 5713 book, and the 5304 book, respectively.

Table 2 on page 6 and Table 3 on page 7 map the OS/400-supported operations given in Table 1 to the high-level language operations (BASIC, ILE C, ILE COBOL, PASCAL, PL/I, and ILE RPG programming languages) supported on the system. For additional information on each operation and how it correlates to the file declaration in the program, see the appropriate language information. Note that not all OS/400 operations are supported in all languages.

Table 1. File Types and Their Main Operations

	File Types							
Operation	**Database**	**Diskette**	**Tape**	**Printer**	**Display**	**ICF**	**DDM**	**Save**
OPEN								
Read	X	X	X	-	X	X	X	X
Write	X	X	X	X	X	X	X	X
Update	X	-	-	-	X[1]	-	X	-
Delete	X	-	-	-	X[1]	-	X	-
READ								
By relative record number	X	-	-	-	X[1]	-	X	-
By key	X	-	-	-	-	-	X	-
Sequential	X	X	X	-	X	X	X	X
Previous	X	-	X	-	-	-	X	-
Next	X	X	X	-	X	X	X	X
Invited								

Table 1. File Types and Their Main Operations (continued)

Operation	File Types							
	Database	**Diskette**	**Tape**	**Printer**	**Display**	**ICF**	**DDM**	**Save**
Device	-	-	-	-	X	X	-	-
WRITE-READ	-	-	-	-	X	X	-	-
WRITE								
By relative record number	X	-	-	-	X[1]	-	X	-
By key	X	-	-	-	-	-	X	-
Sequential	X	X	X	X	X	X	X	X
FEOD	X	X	X	X	-	-	X	X
UPDATE								
By relative record number	X	-	-	-	X[1]	-	X	-
By key	X	-	-	-	-	-	X	-
DELETE								
By relative record number	X	-	-	-	-	-	X	-
By key	X	-	-	-	-	-	X	-
ACQUIRE	-	-	-	-	X	X	-	-
RELEASE	-	-	-	-	X	X	-	-
COMMIT	X	-	-	-	-	-	-	-
ROLLBACK	X	-	-	-	-	-	-	-
CLOSE	X	X	X	X	X	X	X	X

Note:

[1] Operation allowed only for subfile record formats

Table 2. High-Level Languages and Their OS/400 Operations

	High-Level Languages		
Operation	**BASIC**	**ILE C/400 Programming Language**	**ILE COBOL/400 Programming Language**
OPEN			
Read	OPEN INPUT	fopen, _Ropen	OPEN INPUT
Write	OPEN OUTPUT	fopen, _Ropen	OPEN OUTPUT, OPEN EXTEND
Update	OPEN OUTIN	fopen, _Ropen	OPEN I-O
Delete	OPEN OUTIN	fopen, _Ropen	OPEN I-O
READ			
By relative record number	READ REC	_Rreadd	READ
By key	READ KEY	_Rreadk, _Rformat	READ KEY
Sequential	READ NEXT, GET	fread, fgetc, fgets, _Rreadf, _Rreadl, _Rreadn, _Rreadp, _Rreads, _Rformat, _Rpgmdev	READ
Previous	READ PRIOR	_Rreadp	READ
Next	READ NXT, GET	fread, _Rreadn	READ, READ NEXT
Invited Device		_Rreadindv	READ
WRITE-READ		_Rwriterd, _Rformat, _Rpgmdev	
WRITE			
By relative record number	WRITE REC	_Rwrited	WRITE
By key	WRITE	_Rwrite, _Rformat	
Sequential	WRITE	fwrite, fputc, fputs, _Rwrite, _Rformat, _Rpgmdev	WRITE
FEOD		_Rfeod	
UPDATE			
By relative record number	REWRITE REC	_Rupdate	REWRITE
By key	REWRITE KEY	_Rupdate	REWRITE
DELETE			
By relative record number	DELETE REC	_Rdelete	DELETE
By key	DELETE KEY	_Rdelete	DELETE
ACQUIRE		_Racquire	ACQUIRE
RELEASE		_Rrelease	DROP

Table 2. High-Level Languages and Their OS/400 Operations (continued)

Operation	High-Level Languages		
	BASIC	**ILE C/400 Programming Language**	**ILE COBOL/400 Programming Language**
COMMIT		_Rcommit	COMMIT
ROLLBACK			ROLLBACK
CLOSE	CLOSE, END	fclose, _Rclose	CLOSE, STOP RUN, CANCEL

Table 3. High-Level Languages and Their OS/400 Operations

Operation	High-Level Languages		
	PASCAL	**PL/I**	**ILE RPG/400 Programming Language**
OPEN			
Read	RESET, GET, READ, READLN	OPEN INPUT	OPEN
Write	REWRITE, WRITE, WRITELN	OPEN OUTPUT	OPEN
Update	UPDATE	OPEN UPDATE	OPEN
Delete	UPDATE	OPEN UPDATE	OPEN
READ			
By relative record number	GET, READ	READ KEY	READ, CHAIN
By key		READ KEY	READ, READE, CHAIN
Sequential	GET, READ, READLN	READ NEXT, GET	READ
Previous	GET, READ, READLN	READ PRV	READP, READPE
Next	GET, READ, READLN	READ NXT, GET	READ, READE
Invited Device			READ
WRITE-READ			EXFMT
WRITE			
By relative record number	PUT, WRITE, WRITELN	WRITE, EXCPT primary file	WRITE
By key		WRITE KEY	WRITE, EXCEPT
Sequential	PUT, WRITE, WRITELN	WRITE, PUT	WRITE, EXCEPT
FEOD			FEOD
UPDATE			
By relative record number	PUT, WRITE, WRITELN	REWRITE KEY	UPDATE

Table 3. High-Level Languages and Their OS/400 Operations (continued)

	High-Level Languages		
Operation	**PASCAL**	**PL/I**	**ILE RPG/400 Programming Language**
By key		REWRITE KEY	UPDATE
DELETE			
By relative record number		DELETE	DELETE
By key		DELETE KEY	DELETE
ACQUIRE			ACQ
RELEASE			REL
COMMIT	use CL COMMIT	PLICOMMIT subroutine	COMMIT
ROLLBACK	use CL ROLLBACK	PLIROLLBACK subroutine	ROLBK
CLOSE	CLOSE, END	CLOSE, STOP	CLOSE, RETURN

Security Considerations

This section describes some of the file security functions. The topics covered include the authorizations needed to use files and considerations for specifying these authorities when creating a file. For more information about using the security function on the system, see the *Security - Reference*.

Object Authority

The following topics describe the types of authority that can be granted to a user for a file. Also, you can use the SQL GRANT and REVOKE statements to assign and remove these AS/400 authorities to SQL tables, including individual columns within those tables. You can find information about these statements in the SQL Reference book.

Object Operational Authority

Allows you to look at an object description and use the object as determined by your data authorities to the object. Object operational authority is required to:

- Open the file for processing. You must also have read authority to the file. For device files that are not using spooling, you must have object operational and also all data authorities to the device.
- Compile a program which uses the file description.
- Display the file description.
- Delete the file.
- Transfer ownership of the file.
- Grant and revoke authority.

- Change the file description.
- Move or rename the file.

Object Existence Authority

Object existence authority is required to:

- Delete the file.
- Save, restore, and free the storage of the file.
- Transfer ownership of the file.

Object Management Authority

Object management authority is required to:

- Grant and revoke authority. You can grant and revoke only the authority that you already have.
- Change the file description.
- Move or rename the file.
- Refer to a database file from another database file.
- Add triggers to and remove triggers from database files.
- Add referential and unique constraints to database files.
- Remove referential and unique constraints to database files.
- Change the attributes of a database file.
- Change the attributes of a SQL package.

Object Reference Authority

Allows you to refer to a database file from another database file. The operations that you can perform on the referred-to database file are determined by the referring database file.

Object Alter Authority

Allows you to alter the attributes of a database file or SQL package. Object alter authority is required to:

- Add triggers to and remove triggers from database files.
- Add referential and unique constraints to database files.
- Remove referential and unique constraints to database files.
- Change the attributes of a database file.
- Change the attributes of a SQL package.

Data Authorities

You can use data authorities to limit user access to the data in files.

You need the following authorities to perform the associated operations:

Execute
Run a program or locate an object in a library.

Read Open any file for input, compile a program using the file, or display the file description.

Add Add new records to the file.

Update
Open a database file for update.

Delete Open a database file for delete.

For files other than database and save files, the execute, the add, update, and delete authorities are ignored.

Authorities Required for File Operations

Table 4 lists the file object authority required for file functions. Table 5 lists the data authority required for file functions. This is the same information that was presented in the previous two sections, but it is listed by function rather than by authority.

Table 4. Object Authority Required for File Operations

Function	Object Operational	Object Existence	Object Management	Object Reference	Object Alter
Open, I/O, close file[1]	X				
Compile a program using the file description	X				
Display file description	X				
Delete file	X	X			
Save/restore		X			
Transfer ownership	X	X			
Grant/revoke authority	X		X		
Change file description	X		X		
Move file	X		X		
Rename file	X		X		
Replace file	X	X	X		
Refer to another file[2]			X	X	
Add or remove file constraints [3]			X		X
Add or remove triggers [4]			X		X
Change attributes [5]			X		X

:

1 For device files that are not using spooling, you must also have object operational and all data authorities to the device.

2 For database files only.

3 For database files only. Parent files need object management or object reference authority. Dependent files need object management or object alter authority.

4 For database files only. Files need object management or object alter authority.

5 For database files and SQL packages only. Files need object management or object alter authority.

Table 5. Data Authority Required for File Operations

Function	Execute	Read	Add	Update	Delete
Open, I/O, close file[1]		X	X[2]	X[3]	X[3]

Table 5. Data Authority Required for File Operations (continued)

Function	Execute	Read	Add	Update	Delete
Compile a program using the file description		X			
Run a program or locate an object in a library	X				
Display file description		X			
Replace file		X			
Add or remove triggers [4]		X	X[5]	X[6]	X[7]

:

1 For device files that are not using spooling, you must also have object operational and all data authorities to the device.

2 Open for output for database and save files.

3 Open for update or delete for database files.

4 For database files only.

5 Add authority required in addition to Read authority for inserting triggers.

6 Update authority required in addition to Read authority for updating triggers.

7 Delete authority required in addition to Read authority for deleting triggers.

Limiting access to files and data when creating files

Specifying authorities allows you to control access to a file.

Specifying authorities when creating files:

To specify public authority when you create a file, use the AUT parameter on the create command.

What public authority is:

Public authority is authority that is available to any user who does not have specific authority to the file or who is not a member of a group that has specific authority to the file. That is, if the user has specific authority to a file or the user is a member of a group with specific authority, then the public authority is not checked when a user performs an operation to the file. Public authority can be specified as:

- *LIBCRTAUT. All users that do not have specific user or group authority to the file have authority determined by the library in which the file is being created. The library value is specified by the *CRTAUT command to establish a public authority for this library.
- *CHANGE. All users that do not have specific user or group authority to the file have authority to use the file. The *CHANGE value is the default public authority. *CHANGE grants any user object operational and all data authorities.
- *USE. All users that do not have specific user or group authority to the file have authority to use the file. *USE grants any user object operational, execute, and read data authority.
- *EXCLUDE. Only the owner, security officer, users with specific authority, or users who are members of a group with specific authority can change or use the file.
- *ALL. All users that do not have specific user or group authority to the file have all data authorities and all object authorities.

- Authorization list name. An authorization list is a list of users and their authorities. The list allows users and their different authorities to be grouped together.

Specifying or changing authorities on existing files:

To specify or change public authority on an existing file, use the Edit Object Authority (EDTOBJAUT), Grant Object Authority (GRTOBJAUT), or Revoke Object Authority (RVKOBJAUT) commands to grant or revoke the public authority of a file.

For more information about using the security function on the system, see the *Security - Reference*.

Sharing Files

AS/400 data management provides several levels of support for shared files. The system automatically provides the first level of support. By default, the system lets many users and more than one job use one file at the same time. The system allocates the file and its associated resources for each use of the file in such a way that it can prevent conflicting uses. Within the same job, programs can share files if one program opens the same file more than once or if different programs open the same file. Even though the same file is being used, each open operation creates a new path from the program to the data or device, so that each open represents an independent use of the file.

Open data path

A closer level of sharing within a job allows more than one program to share the same path to the data or device. This path, called an **open data path**, is the path through which all of the read and write operations for the file are performed. You can use this level of sharing by specifying the SHARE parameter on the create file, change file, and override file commands. The SHARE parameter allows more than one program to share the file status, positions, and storage area. It can improve performance by reducing the amount of main storage the job needs and by reducing the time it takes to open and close the file. AS/400 bases this level of sharing on two models:

- The **original program model** is the set of functions for compiling source code and creating high-level language programs on the AS/400 system before the Integrated Language Environment (ILE) model was introduced.
- The **ILE model** is the set of constructs and interfaces that provide a common run-time environment and run-time bindable application program interfaces (APIs) for all ILE-conforming high-level languages.

Shared files in the original program model

In the original program model, the SHARE(*YES) parameter lets two or more programs that run in the same job share an open data path (ODP). It connects the program to a file. If not specified otherwise, every time a file is opened a new open data path is built. You can specify that if a file is opened more than once and an open data path is still active for it in the same job, the active ODP for the file can be used with the current open of the file; a new open data path does not have to be created. This reduces the amount of time that is required to open the file after the first opened to open the file after the first open, and the amount of main storage that is required by the job. You must specify SHARE(*YES) for the first open and other opens of the same file to share the open data path. A well-designed (for

performance) application will normally do a shared open on database files that multiple programs will open in the same job. Specifying SHARE(*YES) for other files depends on the application.

Shared files in the ILE model

In the ILE model, shared files are scoped either to the job level or to the activation group level. An **activation group** is a substructure of a run-time job. It consists of system resources (storage for program or procedure variables, commitment definitions, and open files) that are allocated to one or more programs. An activation group is like a miniature job within a job.

Any programs that run in any activation group can share shared files that are scoped to the job level. Only programs that run in the same activation group can share shared files that are scoped to the activation group level.

Sharing files: considerations

Sharing files allows you to have programs within a job interact in ways that would otherwise not be possible. However, you should read the following topics to learn more about the effects of opening, performing read and write operations, and closing shared files:

- "Open Considerations for Files Shared in a Job"
- "I/O Considerations for Files Shared in a Job" on page 14
- "Close Considerations for Files Shared in a Job" on page 14

You should also see the appropriate documentation for all of the file types to understand how this support works, and the rules your programs must follow to use it correctly.

Note: Most high-level language programs process an open or a close operation independent of whether or not the file is being shared. You do not specify that the file is being shared in the high-level language program. You indicate that the file is being shared in the same job through the SHARE parameter. You specify the SHARE parameter only on the create, change, and override file commands. Refer to your appropriate language information for more information.

Open Considerations for Files Shared in a Job

Consider the following points when you open a shared file in the same job by specifying SHARE(*YES).

- You must make sure that when the shared file is opened for the first time in a job, all the open options that are needed for subsequent opens of the file are specified. If the open options specified for subsequent opens of a shared file do not match those specified for the first open of a shared file, an error message is sent to the program. (You can correct this by making changes to your program to remove any incompatible options.)

  For example, PGMA is the first program to open FILE1 in the job and PGMA only needs to read the file. However, PGMA calls PGMB which will delete records from the same shared file. Because PGMB will delete records from the shared file, PGMA will have to open the file as if it, PGMA, is also going to delete records. You can accomplish this by using the correct specifications in the

high-level language. (In order to accomplish this in some high-level languages, you may have to use file operation statements that are never run. See your appropriate language information for more details.)

- Sometimes sharing a file within a job is not possible. For example, one program may need records from a file in arrival sequence, and another program may need the records in keyed sequence. Or, you may use the same file for printing output, but want to produce the output from each program separately. In these situations, you should not share the open data path. You would specify SHARE(*NO) on the override command to ensure that programs do not share the file within the job.
- If debug mode is entered with UPDPROD(*NO) after the first open of a shared file in a production library, subsequent shared opens of the file share the original open data path and allow the file to be changed. To prevent this, specify SHARE(*NO) on the override command before opening files while debugging your program.
- The use of commitment control for the first open of a shared file, requires that all subsequent shared opens also use commitment control.
- If you did not specify a library name in the program or the override command (*LIBL is used), the system assumes that the library list has not changed since the last open of the same shared file with *LIBL specified. If the library list has changed, you should specify the library name on the override command to ensure that you opened the correct file.
- The system processes overrides and program specifications that are specified on the first open of the shared file. Overrides and program specifications specified on subsequent opens, other than those that change the file name or the value specified on the SHARE or LVLCHK parameters on the override command, are ignored.

I/O Considerations for Files Shared in a Job

The system uses the same input/output area for all programs sharing the file, so the order of the operations is sequential regardless of which program does the operation. For example, if Program A is reading records sequentially from a database file and it reads record 1 just before calling Program B, and Program B also reads the file sequentially, Program B reads record 2 with the first read operation. If Program B then ends and Program A reads the next record, it receives record 3. If the programs were not sharing the file, Program A would read record 1 and record 2, and Program B would read record 1.

For device files, the device remains in the same state as the last I/O operation.

For display and ICF files, programs other than the first program that opens the file may acquire more display or program devices or release display or program devices already acquired to the open data path. All programs sharing the file have access to the newly acquired devices, and do not have access to any released devices.

Close Considerations for Files Shared in a Job

The processing done when a program closes a shared file depends on whether other programs currently share the open data path. If there are other programs, the main function that is performed is to detach from the file the program that is requesting the close. For database files, the program also releases any record locks

that it holds.. The program will not be able to use the shared file unless it opens it again. All other programs sharing the file are still attached to the ODP and can perform I/O operations.

If the program closing the file is the last program sharing the file, then the close operation performs all the functions it would if the file had not been opened with the share option. This includes releasing any allocated resources for the file and destroying the open data path.

The function provided by this last close operation is the function that is required for recovering from certain run-time errors. If your application is written to recover from such errors and it uses a shared file, this means that all programs that are attached to the file when the error occurs will have to close the file. This may require returning to previous programs in the call stack and closing the file in each one of those programs.

Allocating File Resources

Resources are those parts of the system that are required by a job or task, including main storage, devices, the processing unit, programs, files, libraries, and folders. When you write a high-level language program, you should be aware of what resources the system has allocated for each file type.

Normally, the system will perform the allocation whenever a requested operation requires it. For example, the system allocates resources for each file that is used in a program when the file is opened.

To ensure that all of the resources that are needed by a program are available before the program is run, you can use the Allocate Object (ALCOBJ) CL command in the job before you run the program. In particular, the ALCOBJ command can allocate database files and most devices.

The following operations are examples of operations that require resource allocation:

- Open
- Acquire
- Starting a program on a remote system

See the following topics for more information:

- “File resource allocation”
- “File resources that must be allocated” on page 16
- “How the system allocates resources” on page 16

File resource allocation

When a high-level language program uses a file, several operations require that the system allocate the resources that are needed to perform that operation. The system generally does this to ensure that multiple users do not use the file in conflicting ways.

For example, the system will not allow you to delete a file while any application program is using it. The system does this by obtaining a lock on the file when it

opens. The delete file operation also attempts to get a lock on the file and is unsuccessful because the program using the file still has the lock from when the file was opened, and the locks conflict.

File resources that must be allocated

The file resources that the system must allocate depend on the type of file and the operation. File resources consist of the following:

- Open
  - For printer and diskette files that are spooled (SPOOL(*YES)), the file resources include the file description, the specified output queue, and storage in the system for the spooled data. Because the data is spooled, the device need not be available.
  - For database files, the file resources consist of the entire file; this includes the file, member, data, and the associated access path.
  - For printer and diskette files that are not spooled (SPOOL(*NO)) as well as for tape files, display files, and some ICF files, the file resources include the file description and the device. For ICF files that use APPC, APPN, or intrasystem communications, the file resources include the file description and the session resources that are associated with the device.
  - For save files, the file resources consist of the entire file, including the file and data.
  - For DDM files, the file resources include the file description and the session resources that are associated with the device.
- Acquire operation

  For display files and ICF files that do not use APPC/APPN or intrasystem communications, the system allocates the device as a resource. For ICF files that use APPC/APPN or intrasystem communications, resources include the session resources that are associated with the device.
- Starting a program on a remote system

  Session resources that are needed for APPC and APPN.

How the system allocates resources

When it allocates resources, the system waits for a predefined time if the resources are not immediately available. If the resources do not become available within the time limit, the system generates an error. If you are using the ALCOBJ command, the command fails. If your program is performing a file operation, that operation fails, and the system sends an error message to the program message queue. You may attempt to use the error handling functions of your high-level language to try the operation again. For example, if an open operation fails because another job is using the device associated with the file, you could retry the open operation a specified number of times, in the hope that the other job would finish with the device and your program would then be able to use it.

The length of time that the system waits when allocating resources is specified on the ALCOBJ command and on the WAITFILE parameter of the CL command used to create the file. If the ALCOBJ command is used prior to running a program, then the value of the WAITFILE parameter does not matter, because the resources will be available.

If your application has error handling procedures for handling device errors occurring on device files, you should specify a value of something other than

*IMMED to allow the system to recover from the error. The allocation of resources requested by your program on an open or acquire operation that allows your program to recover from the error will not be successful until the system recovery procedures have been completed for the device.

The following chart describes the values that are allowed for the WAITFILE parameter:

Values	Definition
*IMMED	This value specifies that no wait time is allowed. An immediate allocation of the file resources is required.
*CLS	The job default wait time is used as the wait time for the file resources to be allocated.
number-of-seconds	Specify the maximum number of seconds that the program is to wait for the file resources to be allocated. Valid values are 1 through 32767 (32 767 seconds).

Opening Files

When you want an application to use a file, you do so by referring to that file by name. The file description for that file will then control how the program and the system will interact.

You have two options regarding how your application program uses the file description:

- You can use the file description as it currently exists. In this case, the system uses the file description as is, without any change.
- You can change some or all of the parameters that are associated with the file description. A change made to a file description can be permanent or temporary. See the appropriate book for the device that you are using for information about permanent changes.

See the following topics for information on how the system handles open files:

- "Scoping of opened files"
- "Opening files using temporary file descriptions" on page 18
- "Open Considerations When Using *LIBL with a DDM File" on page 20
- "Displaying information about open files" on page 22
- "Detecting File Description Changes" on page 21
- "Open and I/O Feedback Area" on page 23

Scoping of opened files

Files that are opened within the user default activation group are scoped to the call level number of the calling program (default). A **call level number** is a unique number that the system assigns to each call stack entry. Files that are opened

within a named activation group are scoped to the activation group level (default). You can change the scope of an open operation by using override commands. For example, you can change the scope of an open operation to the job level. For more information on using overrides to change the scope of an open operation, see “Chapter 3. Using Overrides” on page 33. For information on displaying the scope of existing open operations, see “Displaying information about open files” on page 22.

Opening files using temporary file descriptions

Temporary changes can provide greater flexibility to the application. The system makes temporary changes when the program is first establishing a path to the file by opening the file. Temporary changes can be made in one of two ways:

- By information that is specified within the program itself, and which is passed as parameters on the open operation.
- By using override CL commands in the input stream that is used to set up the run-time environment for the application

The ability to use the first way depends very much on which programming language you used to write the program. Some programming languages do not allow you to control the open process to any great extent. These languages do the open process more or less automatically and control what information gets passed. Other languages allow you to have greater control over the open process.

You can use the second option regardless of which programming language you use. AS/400 provides override CL commands for each file type. By including override commands with the application, you may temporarily change the file description in a file that the program wants to use.

You can use both options together. Information that is contained in the application can change some parameters; an override command can change others. Both can change the same parameter. The operating system follows this order when making temporary changes to a file:

1. The file description provides a base of information.
2. Change information received from the application during the open process is applied first to the base information.
3. Change information found in the override command is applied last. If both the change information from the application and the override change the same information, the override has precedence.

Only the application that makes the changes can see the temporary changes. The file, as seen by another application, remains unchanged. In fact, two applications may use the same file at the same time, and each may change it temporarily according to its needs. Neither application is aware the other has made a temporary change. Figure 2 on page 19 and Figure 3 on page 20 illustrate the permanent and temporary change processes.

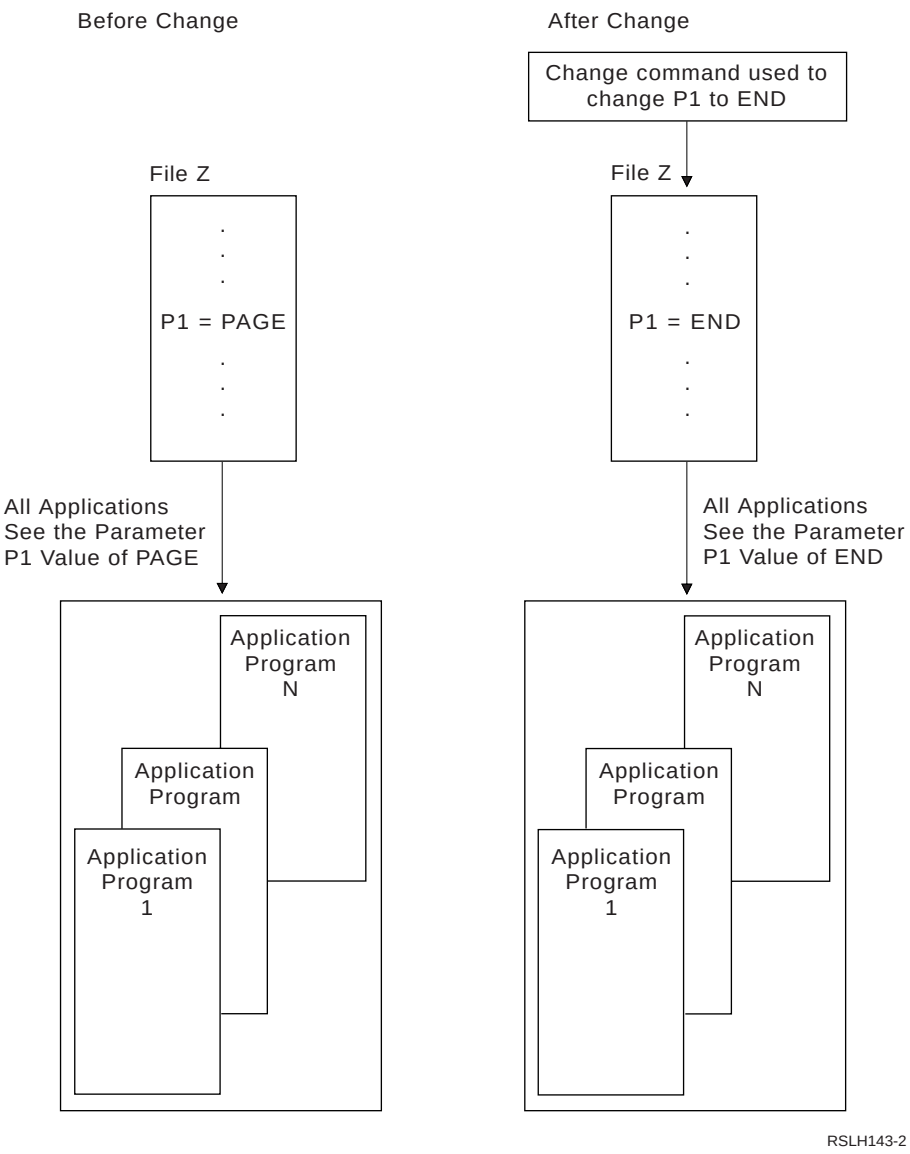


Figure 2. Permanently Changing a File

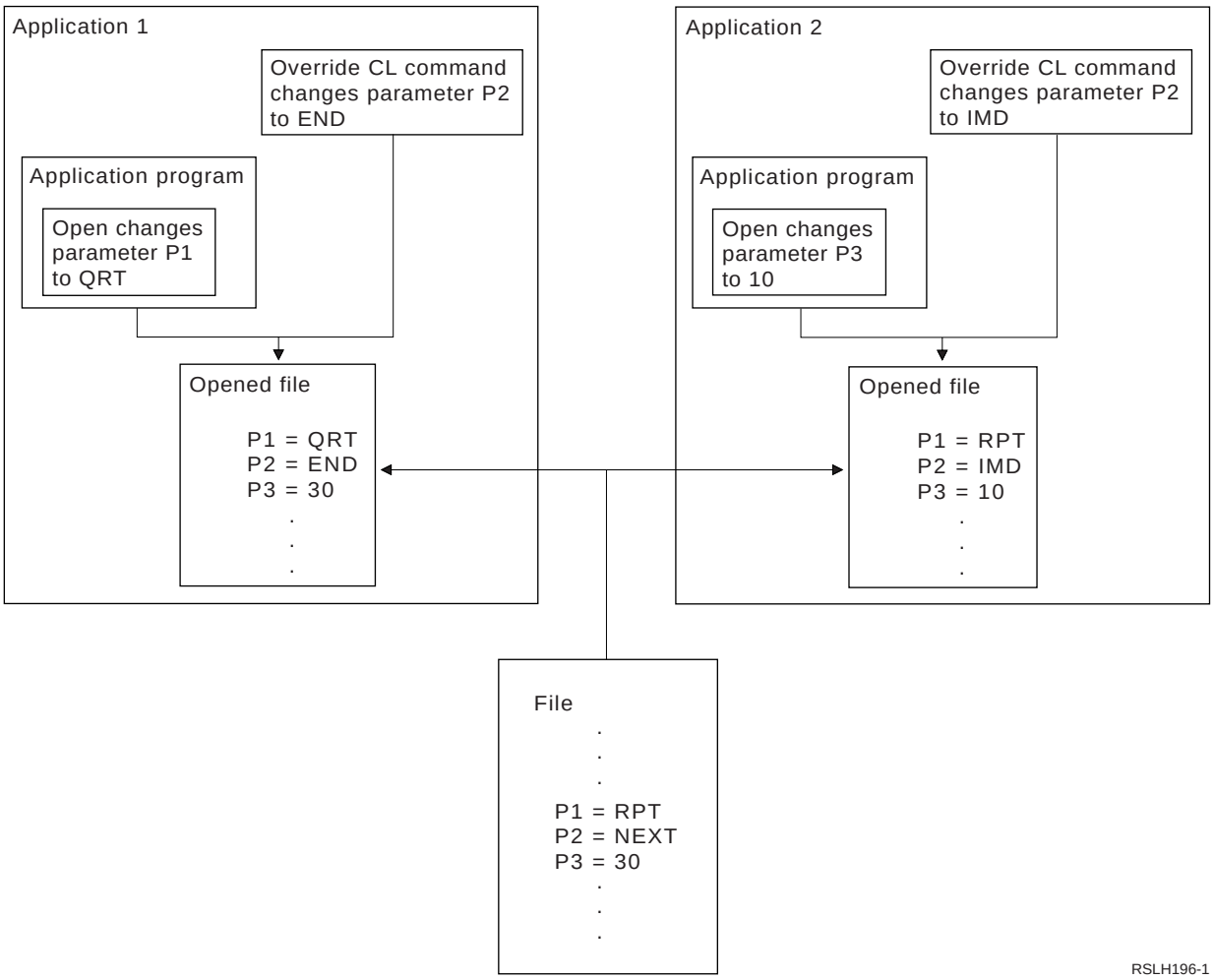


Figure 3. Temporarily Changing a File

Once an application establishes a connection between itself and the file by opening the file, it can then proceed to use the file for either input or output operations. In the case of a database file, the open process establishes a path between the application and the actual database file. For device files, a path is established between the application and the actual device, or to a spooled file if the spooling attribute is active for the device file. In all cases, the application connects to what it wants to use, and those connections determine what input or output operations are valid. Not all operations are valid with all file types. The application must be aware of what file types it uses and then use only those operations which are valid for those types.

Open Considerations When Using *LIBL with a DDM File

Take note of the following considerations when you open DDM files and specify *LIBL for the library:

- The system first searches the library list for a local database file with the specified member. Even if the local database file is located in a library later in your library list than the library containing the DDM file, the local database file containing the specified member is used.

Therefore, if you want to open a DDM file using *LIBL, you must ensure that there are no local database files with the same name, and that contain the specified member, anywhere in your library list.

- If the system does not locate a local database file with the specified member, it searches the library list for the first file that has the specified name. If this file is not of the proper type, or if it does not contain the specified member, an open failure occurs.

  Therefore, if you want to open a DDM file using *LIBL, you must ensure that the DDM file you want to open is the first file in your library list with the specified name.

Detecting File Description Changes

When a program that uses externally described files is compiled, the high-level language compiler extracts the record-level and field-level descriptions for the files referred to in the program and makes those descriptions part of the compiled program. When you run the program, you can verify that the descriptions with which the program was compiled are the current descriptions.

The system assigns a unique level identifier for each record format when it creates the associated file. The system uses the following information to determine the level identifier:

- Record format name
- Field name
- Total length of the record format
- Number of fields in the record format
- Field attributes (for example, length and decimal positions)
- Order of the field in the record format

Note: It is possible for files with large record formats (many fields) to have the same format level identifiers even though their formats may be slightly different. Problems can occur when copying these files if the record format names of the from-file and the to-file are the same.

Display, printer, and ICF files may also use the number of and order of special fields called indicators to determine the level identifier.

If you change the DDS for a record format and change any of the items in the preceding list, the level identifier changes.

To check the record format identifiers when you run the program, specify LVLCHK(*YES) on the create file or change file commands.

The level identifiers of the file opened and the file description that is part of the compiled program are compared when the file is opened and LVLCHK(*YES) is specified. The system does a format-by-format comparison of the level identifiers. If the identifiers differ or if any of the formats specified in the program do not exist in the file, a message is sent to the program to identify the condition.

When the identifiers differ, this means that the file format has changed. If the changes affect a field that your program uses, you must compile the program again for it to run properly. If the changes do not affect the fields that your program uses, you can run the program without compiling again by entering an override command for the file and specifying LVLCHK(*NO). Specifying LVLCHK(*NO) causes the

system to omit the level identifier check when the file opens. For example, suppose that you add a field to the end of a record format in a database file, but the program does not use the new field. You can enter the Override with Database File (OVRDBF) command with LVLCHK(*NO) to enable the program to run without compiling again.

There are several CL commands available to you to check the changes. You can use the Display File Field Description (DSPFFD) command to display the record-level and field-level descriptions or, if you have the source entry utility (SEU), you can display the source file containing the DDS for the file. You can display the format level identifier that is defined in the file by using the Display File Description (DSPFD) or the DSPFFD commands. The format level identifier which was used when the program was created can be displayed by the Display Program References (DSPPGMREF) command.

There are also some changes to a file description that will not cause an error when the file opens. These happen because the record format identifiers did not change or because your program does not use the changed formats. You can add or remove formats from a file without affecting existing programs that do not use the added or deleted formats.

Even though the level identifier does not change, some DDS functions that you add or delete could require changes in the logic of your program. You should review the functions you added or deleted to determine whether the program logic requires changes.

Normally, the use of LVLCHK(*YES) is a good file integrity practice. The use of LVLCHK(*NO) can produce unpredictable results.

Displaying information about open files

You can display information about your open files in two ways:

- Type `dspjob option(*opnf)` on any command line and press Enter.
- Type `wrkjob option(*opnf)` on any command line and press Enter.

The following screen displays:

```
                              Display Open Files

Job . . :   QPADEV0027     User . . :   KELLYMR        Number . . . :   032138
Number of open data paths . . . . . :      2


                                  Member/
File       Library    Device     Scope      Activation Group
QDUI80     QSYS       QPADEV0027 *ACTGRPDFN 0000000002 *DFTACTGRP
QDDSPOF    QSYS       QPADEV0027 *ACTGRPDFN 0000000002 *DFTACTGRP









Press Enter to continue.

F3=Exit   F5=Refresh   F10=Display I/O details   F12=Cancel   F16=Job menu
```

The **Scope** column identifies the level to which the open is scoped. *ACTGRPDFN indicates that the open is scoped to the activation group level. If the file opened in the user default activation group, the open is scoped to the call level number of the calling program. If the file opened in a named activation group, the open is scoped to the activation group level. *JOB indicates that the open is scoped to the job level. You can change the scope of an open operation by using override commands. For information on how to use overrides to change the scope of an open operation, see Chapter 3. Using Overrides.

The **Activation Group** column identifies the number and name of the activation group. *DFTACTGRP indicates the default activation group.

Open and I/O Feedback Area

The system monitors the status of a file in feedback areas once it has successfully opened the file. As the system performs operations on a file, it updates the feedback areas to reflect the latest status. These feedback areas give you greater control over applications and provide important information when errors occur.

The feedback areas are established at open time, and there is one feedback area for each open file. One exception is for shared files, which share feedback areas as well as the data path between the program and the file. For more information on shared opens, see “Sharing Files” on page 12.

Some high-level languages on the system allow you to access the status and other information about the file against which operations are being performed. There are two feedback areas of interest to you:

- Open feedback area

  This area contains information of a general nature about the file after the system has successfully opened the file. Examples include the name and library of the file and the file type. See “Open Feedback Area” on page 153 for a complete list of the information that you can retrieve from the open feedback area. In addition to general information about the file, the open feedback area also contains file-specific information after the system has successfully opened the file. The applicable fields depend on the file type.

The open feedback area also contains information about each device or communications session that is defined for the file.

- Input/output feedback area

  There are two sections of the I/O feedback area that are updated on the successful completion of input and output operations:

  – Common area

    This area contains information about I/O operations that were performed on the file. This includes the number of operations and the last operation performed. See “I/O Feedback Area” on page 163 for a complete list of the information that you can retrieve from the common I/O feedback area.

  – File-dependent feedback area

    This area contains file-specific information for display, database, printer, and ICF files; for example, the major and minor return code and amount of data received from the device. See “I/O Feedback Area for ICF and Display Files” on page 169, “I/O Feedback Area for Printer Files” on page 173, and “I/O Feedback Area for Database Files” on page 173 for a complete list of the information that you can retrieve from the file-dependent I/O feedback area.

The above information areas can be useful to you. For example, when an error occurs with a device file, the program could determine predefined error handling operations based on the major/minor return code in the file-dependent feedback area. If data is being received from a communications device and the application on the other end sends an error, the program could determine that the next operation should be to wait until the next block of data is sent indicating the error. Possibly, the next operation may be to close the file and end the conversation with the application on the other side or wait for the next request from the application.

Another way might include detecting the type of file that actually opened to determine the type of operations that are allowed. If the file type is printer, only output operations are allowed.

Error handling

The system can detect errors when a file is opened, when a program device is acquired or released, during I/O operations to a file, and when the file is closed. When appropriate, the system will automatically try to run a failing operation again, up to a retry limit. When a retry is successful, neither operator nor program action is required.

How the system reports errors:

The system reports errors that can affect the processing of the program in any or all of the following ways:

- A notify, status, diagnostic, or escape message may be sent to the program message queue of the program using the file. These messages may also appear in the job log, depending on the message logging level that is set for the job. See “Messages and Message Monitors” on page 25 for more information.
- The high-level language may return a file status code.
- A major and minor return code is returned in the I/O feedback area for intersystem communications function (ICF), display, and printer files. See “Major and Minor Return Codes” on page 26 for more information.

- A notify, status, diagnostic, or escape message may be sent to the operator message queue (QSYSOPR) or the history message queue (QHST).
- Information regarding the error may be saved in the system error log for use by the problem analysis and resolution programs.
- An alert message may be sent to an operator at another system in the network.
- The normal program flow may be interrupted and control may be transferred to an error-handling subroutine, or other language operations may occur. For additional information about how to handle run-time errors, see the appropriate book for the high-level language.

Only some of these are significant to a program that is attempting error recovery.

Actions to take when you receive an error:

See “Recovering from errors” on page 28 for information on the actions you should take when you receive an error.

Nonrecoverable errors:

Not all file errors allow programmed error recovery. Some errors are permanent; that is, the file, device, or program cannot work until you take some corrective action. This might involve resetting the device by varying it off and on again, or correcting an error in the device configuration or the application program. Some messages and return codes inform the user or the application program of conditions that are information rather than errors, such as change in the status of a communications line, or system action taken for an unexpected condition. In many cases, it is possible for the application program to test for an error condition and take some preplanned recovery action which allows the program to continue without intervention from the operator.

For more information:

The *CL Programming*, SC41-5721-02 book discusses how to use the debug functions to resolve unexpected errors that you encounter in the application programs.

The chapter on handling problems in the *Basic System Operation, Administration, and Problem Handling*, SC41-5206-03 book describes the programs that are available for analyzing and reporting system errors and hardware failures.

Messages and Message Monitors

Displayed messages are the primary source of information for an operator or a programmer who is testing a new application. A message usually contains more specific information than the file status code, the indicators, and the major and minor return code. The control language lets you monitor messages so that the CL program can intercept a message and take corrective action. See the *CL Programming*, SC41-5721-02 book for more information about message types and message monitors. In most high-level languages, the file status code and return codes (which are described in the following section) are more convenient sources of information.

Message numbers are assigned in categories to make it easier for a program to monitor for a group of related messages. Table 6 on page 26 shows the message number ranges that are assigned for file error messages.

Table 6. OS/400 Data Management Message Number Ranges

Message IDs	Operation	Message Type
CPF4001-40FF	Open	Diagnostic and status.
CPF4101-43FF	Open	Escapes that make the file unusable.
CPF4401-44FF	Close	Diagnostic and status.
CPF4501-46FF	Close	Escapes that make the file unusable.
CPF4701-48FF	I/O, Acquire, and Release	Notify with a default reply of cancel, status and escapes that do not make the file or device unusable.
CPF4901-49FF	I/O, Acquire, and Release	Notify with a default reply of ignore or go.
CPF5001-50FF	I/O, Acquire, and Release	Notify with a default reply of cancel.
CPF5101-53FF	I/O, Acquire, and Release	Escapes that make the file or device unusable.
CPF5501-56FF	I/O, Acquire, and Release	Escapes that make the file or device unusable.

Some status messages, CPF4018 for example, are preceded by a diagnostic message that provides additional information. Diagnostic messages may be kept in the job log, depending on the message logging level of the job. If a CL program monitors for CPF4018, CPF5041, or similar messages, it can retrieve the accompanying diagnostic message from the program message queue.

If an error occurs for which an escape message is issued and the message is not monitored, your program will be ended and the message displayed for the operator. You can also monitor status messages, but if you do not monitor them the program continues. Most high-level languages except CL monitor for all the file errors that you are likely to encounter, and provide some standard recovery. Depending on the severity of the error, the high-level language may simply end the program and issue a message of its own. Alternatively, the application programmer may code an error recovery routine to handle errors that are anticipated in that particular application.

Within these error-handling routines, it is usually necessary to examine the file status or major and minor return codes to determine the cause of the error. The books for the language you are using explain how to access file status and major and minor return codes. The information for each language also explains the file status codes as each language defines them.

Major and Minor Return Codes

Major and minor return codes report errors and certain status information for ICF, display, and printer files. They are not used for other files. They usually appear as four characters: the first two refer to the major code and the second two refer to the

minor code. The major code indicates the general type of error, and the minor provides further detail. Minor codes, except zero, have the same or a similar meaning, regardless of the major code with which they are combined.

The application program can test the return code after each I/O operation. If the major return code is 00, the operation completed successfully and the minor return code contains status information that indicates whether a read or a write operation should be performed next. A major return code of 04 or higher indicates that an error occurred. The program may test for any specific errors for which it will attempt programmed recovery. The application program may test for a specific condition by comparing the major and minor codes as a unit, or may identify a class of conditions by testing the major code alone.

Most major and minor return codes are accompanied by any one of several message numbers, for which the typical recovery action is similar. The individual languages file status codes; they may set based on the major and minor return codes.

Table 7 defines the major return codes. See the *Application Display Programming*, SC41-5715-00 book for specific definitions of the major and minor return codes as they are used for display files and the message numbers associated with each. Similar specific definitions for printer files and each of the communications types valid on an ICF file can be found in the *Printer Device Programming*, SC41-5713-03 book and the books for each communications type.

Table 7. Major Return Code Definitions

Code	Definition
00	The operation requested by your program completed successfully. The minor includes state information, such as change direction.
02	Input operation completed successfully, but job is being ended (controlled). The minor includes state information.
03	Successful input operation, but no data was received. The minor includes state information.
04	Error occurred because an output operation was attempted while data was waiting to be read.
08	An acquire operation failed because the device has already been acquired or the session has already been established.
11	A read-from-invited-program-devices operation failed because no device or session was invited.
34	An input exception occurred. The data length or record format was not acceptable for the program.
80	A permanent (unrecoverable) system or file error occurred. Programmer action is required to correct the problem.
81	A permanent (unrecoverable) device or session error occurred during an I/O operation.

Table 7. Major Return Code Definitions (continued)

Code	Definition
82	A device or session error occurred during an open or acquire operation. Recovery may be possible.
83	A device or session error occurred during an I/O operation. Recovery may be possible.

Recovering from errors

The following topics describe the actions you should take to recover from errors that you receive. “Major and Minor Return Codes” on page 26 describes return codes.

Normal Completion

A major and minor return code of 0000 indicates that the operation requested by your program completed successfully. Most of the time, the system issues no message. In some cases, the system might use a diagnostic message to inform the user of some unusual condition that it could not handle, but which might be considered an error under some conditions. For example, it might ignore a parameter that is not valid, or it might take some default action.

For communications devices, a major return code of 00, indicating successful completion with data received, is accompanied by a minor return code that indicates what operation the application program is expected to perform next. The nonzero minor does not indicate an error. No message is issued.

Completion with Exceptions

The system assigns several rather specific major return codes to conditions for which a specific response from the application program is appropriate.

A major return code of 02 indicates that the requested input operation completed successfully, but the system is ending the job in a controlled. The application program should complete its processing as quickly as possible. The controlled cancel is intended to allow programs time to end in an orderly manner. If your program does not end within the time specified on the ENDJOB command, the system will end the job without further notice.

A major return code of 03 indicates that an input operation completed successfully without transferring any data. For some applications, this might be an error condition, or it might be expected when the user presses a function key instead of entering data. It might also indicate that all the data has been processed, and the application program should proceed with its completion processing. In any case, the contents of the input buffer in the program should be ignored.

A major and minor code of 0309 indicates that the system received no data *and* is ending the job in a controlled manner. A major and minor code of 0310 indicates that there is no data because the specified wait time has ended. Other minor return codes accompanying the 02 or 03 major code are the same as for a 00 major code, indicating communications status and the operation to be performed next.

A major return code of 04 indicates that an output exception occurred. Specifically, your program attempted to send data when data should have been received. This is probably the result of not handling the minor return code properly on the previous

successful completion. Your program can recover by simply receiving the incoming data and then repeating the write operation.

A major return code of 34 indicates that an input exception occurred. The received data was either too long or incompatible with the record format. The minor return code indicates what was wrong with the received data, and whether the data was truncated or rejected. Your program can probably handle the exception and continue. If the data was rejected, you may be able to read it by specifying a different record format.

Two other return codes in this group, 0800 and 1100, are both usually the result of application programming errors, but are still recoverable. 0800 indicates that an acquire operation failed because the device has already been acquired or the session has already been established. 1100 indicates that the program attempted to read from invited devices with no devices invited. In both cases, the program ignored the request that is not valid, and the program may continue.

No message is issued with a 02 major code or most minor codes with the 03 major code, but the other exceptions in this group are usually accompanied by a message in the CPF4701-CPF47FF or CPF5001-CPF50FF range.

Permanent System or File Error

A major return code of 80 indicates a serious error that affects the file. The application program must close the file and reopen it before attempting to use it again, but recovery is unlikely until the problem causing the error is found and corrected. To reset an error condition in a shared file by closing it and opening it again, all programs sharing the open data path must close the file. This may require returning to previous programs in the call stack and closing the shared file in each of those programs. The operator or programmer should refer to the text of the accompanying message to determine what action is appropriate for the particular error.

Within this group, several minor return codes are of particular interest. A major and minor code of 8081 indicates a serious system error that probably requires an APAR. The message sent with the major and minor return code may direct you to run the Analyze Problem (ANZPRB) command to obtain more information.

A major and minor code of 80EB indicates that incorrect or incompatible options were specified in the device file or as parameters on the open operation. In most cases you can close the file, end the program, correct the parameter that is not valid with an override command, and run the program again. The override command affects only the job in which it is issued. It allows you to test the change easily, but you may eventually want to change or re-create the device file as appropriate to make the change permanent.

Permanent Device or Session Error on I/O Operation

A major return code of 81 indicates a serious error that affects the device or session. This includes hardware failures that affect the device, communications line, or communications controller. It also includes errors due to a device being disconnected or powered off unexpectedly and abnormal conditions that were discovered by the device and reported back to the system. Both the minor return code and the accompanying message provide more specific information regarding the cause of the problem.

Depending on the file type, the program must either close the file and open it again, release the device and acquire it again, or acquire the session again. To reset an error condition in a shared file by closing it and opening it again, all programs sharing the open data path must close the file. In some cases, the message may instruct you to reset the device by varying it off and on again. It is unlikely that the program will be able to use the failing device until the problem causing the error is found and corrected, but recovery within the program may be possible if an alternate device is available.

Some of the minor return codes in this group are the same as those for the 82 major return code. Device failures or line failures may occur at any time, but an 81 major code occurs on an I/O operation. This means that your program had already established a link with the device or session. Therefore, the program may transfer some data, but when the program starts from the beginning when it starts again. A possible duplication of data could result.

Message numbers accompanying an 81 major code may be in the range that indicates either an I/O or a close operation. A device failure on a close operation simply may be the result of a failure in sending the final block of data, rather than action specific to closing the file. An error on a close operation can cause a file to not close completely. Your error recovery program should respond to close failures with a second close operation. The second close will always complete, regardless of errors.

Device or Session Error on Open or Acquire Operation

A major return code of 82 indicates that a device error or a session error occurred during an open or acquire operation. Both the minor return code and the accompanying message will provide more specific information regarding the cause of the problem.

Some of the minor return codes in this group are the same as those for the 81 major return code. Device or line failures may occur at any time, but an 82 major code indicates that the device or session was unusable when your program first attempted to use it. Thus no data was transferred. The problem may be the result of a configuration or installation error.

Depending on the minor return code, it may be appropriate for your program to recover from the error and try the failing operation again after some waiting period. You should specify the number of times you try in your program. It may also be possible to use an alternate or backup device or session instead.

Message numbers accompanying an 82 major code may be in the range indicating either an open or an acquire operation. If the operation was an open, it is necessary to close the partially opened file and reopen it to recover from the error. If the operation was an acquire, it may be necessary to do a release operation before trying the acquire again. In either case, you should specify a wait time for the file that is long enough to allow the system to recover from the error.

Recoverable Device or Session Errors on I/O Operation

A major return code of 83 indicates that an error occurred in sending data to a device or receiving data from the device. Recovery by the application program is possible. Both the minor return code and the accompanying message provide more specific information regarding the cause of the problem.

Most of the errors in this group are the result of sending commands or data that are not valid to the device, or sending valid data at the wrong time or to a device that is not able to handle it. The application program may recover by skipping the failing operation or data item and going on to the next one, or by substituting an appropriate default. There may be a logic error in the application.

Related Information on File Types

Refer to the following books for more information on the file types discussed in this chapter:

- Database files: *DB2 UDB for AS/400 Database Programming*, SC41-5701-02
- Display files: *Application Display Programming*, SC41-5715-00
- DDM files: *Distributed Data Management*, SC41-5307-00
- ICF files: *ICF Programming*, SC41-5442-00
- Printer files: *Printer Device Programming*, SC41-5713-03
- Save files: *Backup and Recovery*
- Tape and diskette files: *Tape and Diskette Device Programming*, SC41-5716-01

Chapter 3. Using Overrides

This topic describes how you use overrides on AS/400.

Overrides

An **override** is a CL command that temporarily changes a file name, a device name, or remote location name associated with the file, or some of the other attributes of a file. You can enter override commands interactively from a display station or submit them as part of a batch job. You can include them in a control language (CL) program, or issue them from other programs by calling the program QCMDEXC. Regardless of how they are issued, overrides remain in effect only for the job, program, or display station session in which they are issued. Furthermore, they have no effect on other jobs that may be running at the same time.

When you create an application program, the file names specified in the program associate files with it. The system lets you override these file names or the attributes of the specified file when you compile a program or run a program.

You can use overrides to change most, but not all, of the file attributes that are specified when the file is created. In some cases, you can specify attributes in overrides that are not part of the original file definition. Refer to the command descriptions in the *CL Reference (Abridged)* for details.

Overriding a file is different from changing a file in that an override does not permanently change the attributes of a file. For example, if you override the number of copies for a printer file by requesting six copies instead of two, the file description for the printer file still specifies two copies, but six copies are printed. The system uses the file override command to determine which file to open and what its file attributes are.

How you work with overrides:

The system supplies three override functions:

- “Applying overrides” on page 36
- “Deleting overrides” on page 55
- “Displaying overrides” on page 58

For additional information:

“Benefits of using overrides” on page 34 provides information about the types of situations where overrides can be especially useful.

“Summary of the override commands” on page 34 provides a list of the commands that you can use to work with overrides.

“Effect of overrides on some commands” on page 35 provides information about how the override commands interact with other system functions.

Handling overrides for message files is different in some respects from handling overrides for other files. You can override only the name of the message file, and not the attributes. For more information on message handling, refer to the *CL Programming* book.

Benefits of using overrides

Overrides are particularly useful for making minor changes to the way a program functions or for selecting the data on which it operates without having to recompile the program. Their principal value is in allowing you to use general purpose programs in a wider variety of circumstances. Examples of items where you can use overrides include the following:

- Changing the name of the file to process
- Selecting the database file member to process
- Indicating whether to spool output
- Directing output to a different tape unit
- Changing printer characteristics such as lines per inch and number of copies
- Selecting the remote location to use with an ICF file
- Changing the characteristics of a communications session

Summary of the override commands

You can process override functions for files by using the following CL commands:

DLTOVR
The Delete Override command deletes one or more file overrides, including overrides for message files, that were previously specified in a call level.

DSPOVR
The Display Override command displays file overrides at any active call level, activation group level, or job level for a job.

OVRDBF
The Override with Database File command iverrides (replaces) the database file named in the program, overrides certain parameters of a database file that is used by the program, or overrides the file and certain parameters of the file to be processed.

OVRDKTF
The Override with Diskette File command overrides (replaces) the diskette file named in the program, overrides certain parameters of a diskette file that is used by the program, or overrides the file and certain parameters of the file to be processed.

OVRDSPF
The Override with Display File command overrides (replaces) the display file named in the program, overrides certain parameters of a display file that is used by the program, or overrides the file and certain parameters of the file to be processed.

OVRICFF
The Override with Intersystem Communications Function File command overrides the file that is named in the program, and overrides certain parameters of the processed file.

OVRMSGF
The Override with Message File command overrides a message file that is used in a program. The rules for applying the overrides in this command are different from the other override commands. For more information on overriding message files, see the *CL Programming* book.

OVRPRTF
The Override with Printer File command overrides (replaces) the printer file

named in the program, overrides certain parameters of a printer file that is used by the program, or overrides the file and certain parameters of the file to be processed.

OVRSAVF
The Override with Save File command overrides (replaces) the file named in the program, overrides certain attributes of a file that is used by the program, or overrides the file and certain attributes of the file to be processed.

OVRTAPF
The Override with Tape File command overrides (replaces) the file named in the program, overrides certain attributes of a file that is used by the program, or overrides the file and certain attributes of the file to be processed.

Effect of overrides on some commands

The following commonly used commands ignore overrides entirely:

ADDLFM
ADDPFM
ALCOBJ
APYJRNCHG
CHGOBJOWN
CHGPTR
CHGSBSD
CHGXXXF (all change file commands)
CLRPFM
CLRSAVF
CPYIGCTBL
CRTDKTF
CRTDUPOBJ
CRTAUTHLR
CRTSBSD
CRTTAPF
DLCOBJ
DLTF
DLTAUTHLR
DSPDBR
DSPFD
DSPFFD
DSPJRN
EDTOBJAUT
EDTDLOAUT
ENDJRNPF
GRTOBJAUT
INZPFM
MOVOBJ
RGZPFM
RMVJRNCHG
RMVM
RNMOBJ
RTVMBRD
RVKOBJAUT
SBMDBJOB
SIGNOFF
STRDBRDR
STRJRNPF

Note: Save operations and restore operations ignore all file overrides that are related to the respective media (tape, diskette, save file).

The system does not apply overrides to any system files that are opened as part of an end-of-routing step or end-of-job processing. For example, you cannot specify overrides for the job log file. In some cases, when you need to override something in a system file, you may be able to change it through a command other than an override command. For example, to change the output queue for a job log, the output queue could be changed before sign-off using the OUTQ parameter on the Change Job (CHGJOB) command to specify the name of the output queue for the job. If the printer file for the job log contains the value *JOB for the output queue, the output queue is the one that is specified for the job.

The following commands allow overrides for the SRCFILE and SRCMBR parameters only:

CRTCMD
CRTICFF
CRTDSPF
CRTLF
CRTXXXPGM
CRTPRTF
CRTSRCPF
CRTTBL
CRTPF

(All create program commands. These commands also use overrides to determine which file will be opened by a compiled program. See “Applying overrides when compiling a program” on page 54 for more information.)

The following command allows overrides for the TOFILE, MBR, SEQONLY, LVLCHK, and INHWRT parameters:

OPNQRYF

The following commands allow overrides, but do not allow changing the MBR to *ALL:

CPYFRMPCD
CPYTOPCD

The following commands do not allow overrides to affect the display files that they use. Overrides to the printer files they use should not change the file type or the file name. Some restrictions are placed on changes that may be made to printer files used by these commands, but the system can not guarantee that all combinations of possible specifications will produce an acceptable report.

DMPOBJ and DMPSYSOBJ
(In addition to the preceding limitations, these commands do not allow overrides to the file they dump.)

DSPXXXXXX
(All display commands. The display commands that display information about a file do not allow overrides to that file.)

DSPIGCDCT

EDTIGCDCT

GO (You can override message files.)

PRTXXXXXX
(All print commands.)

QRYDTA

TRCXXX
(All trace commands.)

WRKXXXXXX
(All work-with commands.)

Applying overrides

You can perform two general types of overrides:

- **File overrides**

  File overrides let you override the following things:
  - File attributes
  - File names
  - File attributes and file names simultaneously

- File open scope
- File types

  For more information on overriding file types, see “Redirecting files” on page 65.

- **Overrides for program device entries**

  Overrides for program device entries let you override the attribute of an ICF file that provides the link between the application and each of the remote systems or devices with which your program communicates. For more information on overrides on program device entries, see the *ICF Programming* book.

How to apply overrides:

The following scenarios provide detailed examples of how you perform each of the override types:

- “Overriding file attributes”
- “Overriding file names” on page 39
- “Overriding file names and file attributes” on page 39
- “Overriding the scope of an open file” on page 40

For additional information:

The following topics provide additional information about how overrides work on AS/400 and how they affect and are affected by different events:

- “How the system processes overrides” on page 40
- “Effect of exits on overrides: scenario” on page 48
- “Effect of TFRCTL on overrides-Scenario” on page 49
- “Overrides to the same file at the same call level: scenario 1” on page 50
- “Overrides to the same file at the same call level: scenario 2” on page 50
- “CL program overrides” on page 51
- “Securing files against overrides” on page 51
- “Using a generic override for printer files” on page 52
- “Applying overrides when compiling a program” on page 54

Overriding file attributes

The simplest form of overriding a file is to override some attributes of the file. File attributes are built as a result of the following:

- Create file and add member commands. Initially, these commands build the file attributes.
- Program using the files. At compile time, the user program can specify some of the file attributes. (The attributes that you can specify depend on the high-level language in which the program is written.)
- Override commands. At the time when a program runs, these commands can override the file attributes previously built by the merging of the file description and the file parameters specified in the user program.

For example, assume that you create a printer file OUTPUT whose attributes are:

- Page size of 60 by 80
- Six lines per inch
- Two copies of printed output
- Two pages for file separators
- Overflow line number of 55

The Create Printer File (CRTPRTF) command looks like this:

```
CRTPRTF FILE(QGPL/OUTPUT) SPOOL(*YES) +
  PAGESIZE(60 80) LPI(6) COPIES(2) +
  FILESEP(2) OVRFLW(55)
```

You specify the printer file OUTPUT in your application program with an overflow line number of 58 and a page size of 66 by 132.

However, before you run the application program, you want to change the number of printed copies to 3, and the overflow line to 60. The override command looks like this:

```
OVRPRTF FILE(OUTPUT) COPIES(3) OVRFLW(60)
```

Then you call the application program, and three copies of the output print.

When the application program opens the OUTPUT file, the system merges the file-specified attributes, program-specified attributes, and override-specified attributes to form the open data path. The system uses the open data path when the program runs. The system merges file-specified overrides with the program-specified attributes first. Then it merges these merged attributes with the override attributes. In this example, when the OUTPUT file is opened and output operations are performed, spooled output will be produced with a page size of 66 by 132, six lines per inch, three copies, two file separator pages, and overflow at 60 lines.

Figure 4 on page 39 explains this example.

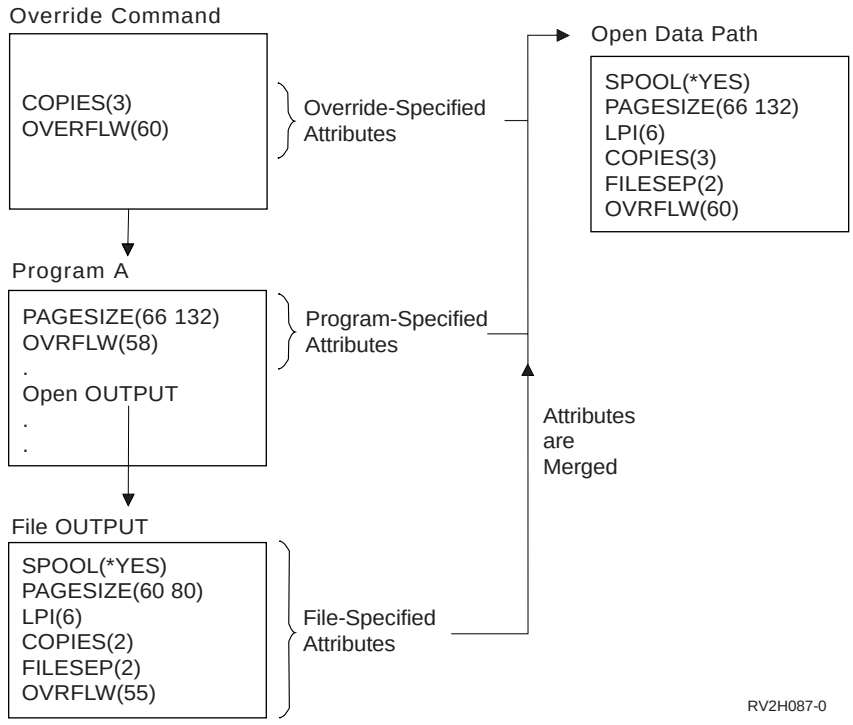


Figure 4. Overriding File Attributes

Overriding file names

Another simple form of overriding a file is to change the file that is used by the program. This may be useful for files that you moved or renamed after the program compiled.

For example, you want to print the output from your application program by using the printer file REPORTS instead of the printer file OUTPUT (the application program specifies the OUTPUT printer file). Before you run the program, enter the following:

```
OVRPRTF FILE(OUTPUT) TOFILE(REPORTS)
```

The CRTPRTF command must have created the file REPORTS before it can use the file.

Overriding file names and file attributes

This form of overriding files is simply a combination of overriding file attributes and overriding file names. With this form of override, you can override the file that is to be used in a program and you can also override the attributes of the overriding file. For example, you want the output from your application program to print using the printer file REPORTS instead of the printer file OUTPUT (the application program specifies the OUTPUT printer file). In addition to having the application program use the printer file REPORTS, you want to produce three copies. Assume that the following command created the file REPORTS:

```
CRTPRTF FILE(REPORTS) SPOOL(*YES) +
  PAGESIZE(68 132) LPI(8) OVRFLW(60) +
  COPIES(2) FILESEP(1)
```

Before you run the program, type the following command:

```
OVRPRTF FILE(OUTPUT) TOFILE(REPORTS) COPIES(3)
```

Then call the application program, and the program produces three copies of the output using the printer file REPORTS.

Note that this is *not* equal to the following two override commands:

```
Override 1 OVRPRTF FILE(OUTPUT) TOFILE(REPORTS)
Override 2 OVRPRTF FILE(REPORTS) COPIES(3)
```

Only one override is applied for each call level for an open of a particular file; therefore, if you want to override the file that the program uses and also override the attributes of the overriding file from one call level, you must use a single command. If you use two overrides, the first override uses the printer file REPORTS to print the output. The system ignores the second override.

Overriding the scope of an open file

To change the scope of a file open operation, use the open scope (OPNSCOPE) parameter on the appropriate override command. The values for the OPNSCOPE parameter can be either *JOB or *ACTGRPDFN (default). Use this parameter to change the scope of an open operation from the call level number or activation group level to the job level.

For example, the following override command scopes the open operation of the BILLING file to the job level:

```
 OVRDBF FILE(BILLING) OPNSCOPE(*JOB)
```

How the system processes overrides

Figure 5 on page 41 shows a representation of a job running in the integrated language environment.

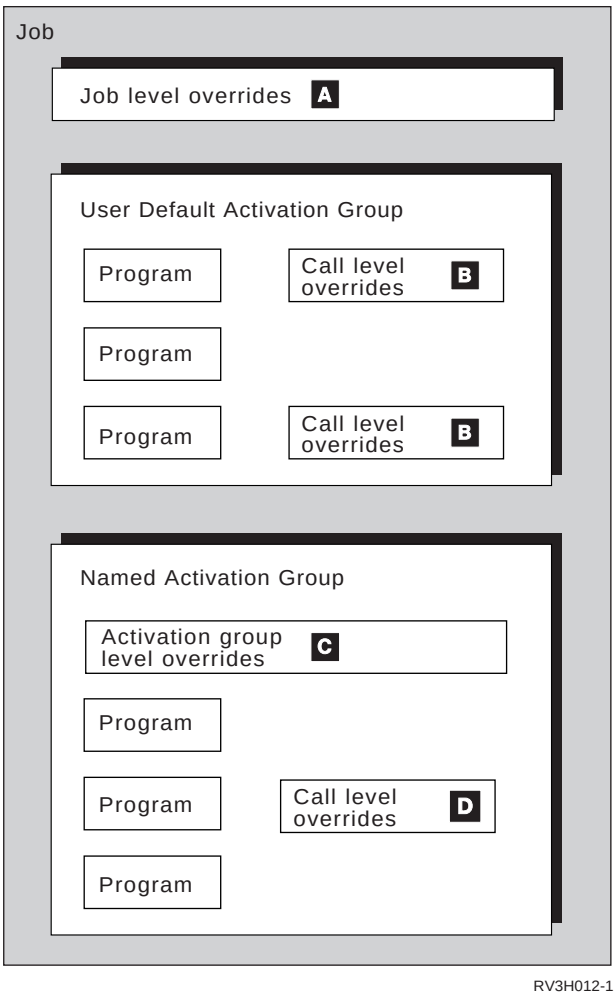


Figure 5. A Job in the Integrated Language Environment

In the description that follows, the reference keys refer to the corresponding reference keys in Figure 5.

In the integrated language environment, overrides can be scoped to the call level, the activation-group level (the default), and the job level. A **job** is a piece of work that the system performs. An **interactive job** begins when a user signs on and ends when a user signs off. Overrides (**A**) that are scoped to the job level have affect on all programs that are running in any activation group within the job. There can be only one active override for a file at the job level. If you specify more than one, the most recent one takes effect. An override that is scoped to the job level remains in effect until one of the following occurs:

- The job ends
- The system explicitly deletes the override
- Another job level override for the same file replaces the override

This is true regardless of the call level in which the overrides were specified. For example, an override that is issued in call level 3 that is scoped to the job level remains in effect when call level 3 is deleted. Overrides can be scoped to the job level by specifying OVRSCOPE(*JOB) on the override command.

Overrides (**B**) that are specified in the user default activation group can be scoped to the call level or to the job level. They cannot be scoped to the user default activation group level. However, overrides (**C** and **D**) that are specified in a named activation group can be scoped to the call level, activation group level, or the job level. Overrides (**C**) scoped to a named activation group level remain in effect until the system replaces or deletes the override, or until the system deletes the named activation group.

Overrides (**D**) that are scoped to the call level within a named activation group remain in effect until they are replaced, deleted, or until the program in which they were issued ends. Overrides can be scoped to the call level by specifying OVRSCOPE(*CALLLVL) on the override command.

Overrides that are scoped to a named activation group level apply only to programs that run in the named activation group. They have no affect on programs that run in other named activation groups or in the user default activation group.

Call levels identify the subordinate relationships between related programs when one program calls another program within a job. Overrides that are scoped to the call level remain in effect from the time they are specified until they are replaced, or deleted, or until the program in which they are specified ends. This is true whether you issue the override in the user default activation group or in a named activation group.

For example:

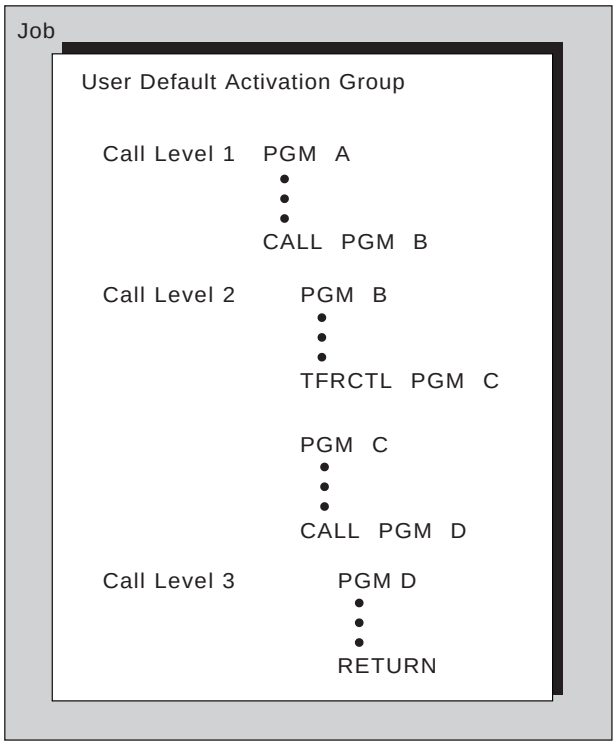


Figure 6. Call Levels within a Job

Several commands, such as Work with Job (WRKJOB), Work with Active Jobs (WRKACTJOB), or Display Job (DSPJOB), have options that allow you to display the call stack of an active job. There is a one-to-one relationship between a

program that is displayed in the call stack and the call level. The first program name displayed (at the top of the list) on the call stack is the program at call level 1 for that job. Call level 1 is the lowest call level for a job. The second program name displayed is the program at call level 2 for that job. The last program name displayed is the program at the highest call level for that job.

In the example in Figure 6 on page 42, the transfer control (TFRCTL) command to PGMC causes PGMC to replace PGMB from the call stack. A CALL command places another program in the call stack. A RETURN command removes a program from the stack.

Processing priority of overrides

The system processes overrides when an open operation occurs in the following order:

1. Call level overrides up to and including the level of the oldest procedure in the activation group are applied first.
2. Activation group level overrides that were specified within the same activation group that the open operation was issued are applied.
3. Call level overrides below the level of the oldest procedure in the activation group are applied.
4. Job level overrides are applied.

How the system processes overrides-scenario 1

The following shows an example of how overrides work in multiple activation groups:

```
                Program A (in user default activation group)
Call Level 2    OVRPTRF FILE(YYY) FOLD(*YES) OVRSCOPE(*CALLLVL)
                CALL PGM B

Call Level 3      Program B (in activation group 8)
                  OVRPTRF FILE(ZZZ) TOFILE(YYY) DEV(P1) LPI(6) +
                    OVRSCOPE(*CALLLVL)
                  CALL PGM C

Call Level 4        Program C (in user default activation group)
                    OVRPTRF FILE(ZZZ) CPI(12) OVRSCOPE(*CALLLVL)
                    CALL PGM D

Call Level 5          Program D (in activation group 21)
                      OVRPTRF FILE(YYY) DEV(P2) OVRSCOPE(*JOB)
                      CALL PGM E

Call Level 6            Program E (in activation group 21)
                        OVRPTRF FILE(ZZZ) LPI(12) OVRSCOPE(*ACTGRPDFN)
                        CALL PGM F

Call Level 7              Program F (in activation group 8)
                          OVRPTRF FILE(ZZZ) LPI(9) OVRSCOPE(*CALLLVL)
                          CALL PGM G

Call Level 8                Program G (in activation group 8)
                            OVRPTRF FILE(ZZZ) DUPLEX(*NO) +
                              OVRSCOPE(*ACTGRPDFN)
                            CALL PGM H

Call Level 9                  Program H (in activation group 8)
                              OVRPTRF FILE(YYY) LPI(5) OVRSCOPE(*ACTGRPDFN)
                              CALL PGM I

Call Level 10                   Program I (in activation group 8)
                                OPEN FILE(ZZZ)
```

Figure 7. Example of Override Processing in Multiple Activation Groups

When program I opens file ZZZ, file ZZZ has the following attributes:

CPI(12)
From call level 4

FILE(YYY)
From call level 3

LPI(5) From call level 9

FOLD(*YES)
From call level 2

DEV(P2)
From call level 5

The system processes the overrides in the following order:

1. File ZZZ opens at call level 10. The system looks for any overrides issued at call level 10 that were scoped to the call level. There are no such overrides.
2. The system searches the next previous call level (level 9) for applicable overrides that were scoped to the call level. There are no such overrides. (The override issued in call level 9 is for file YYY and does not apply.)
3. The system searches call level 8 for applicable overrides that were scoped to the call level. There is an override for file ZZZ; however, it is scoped to the

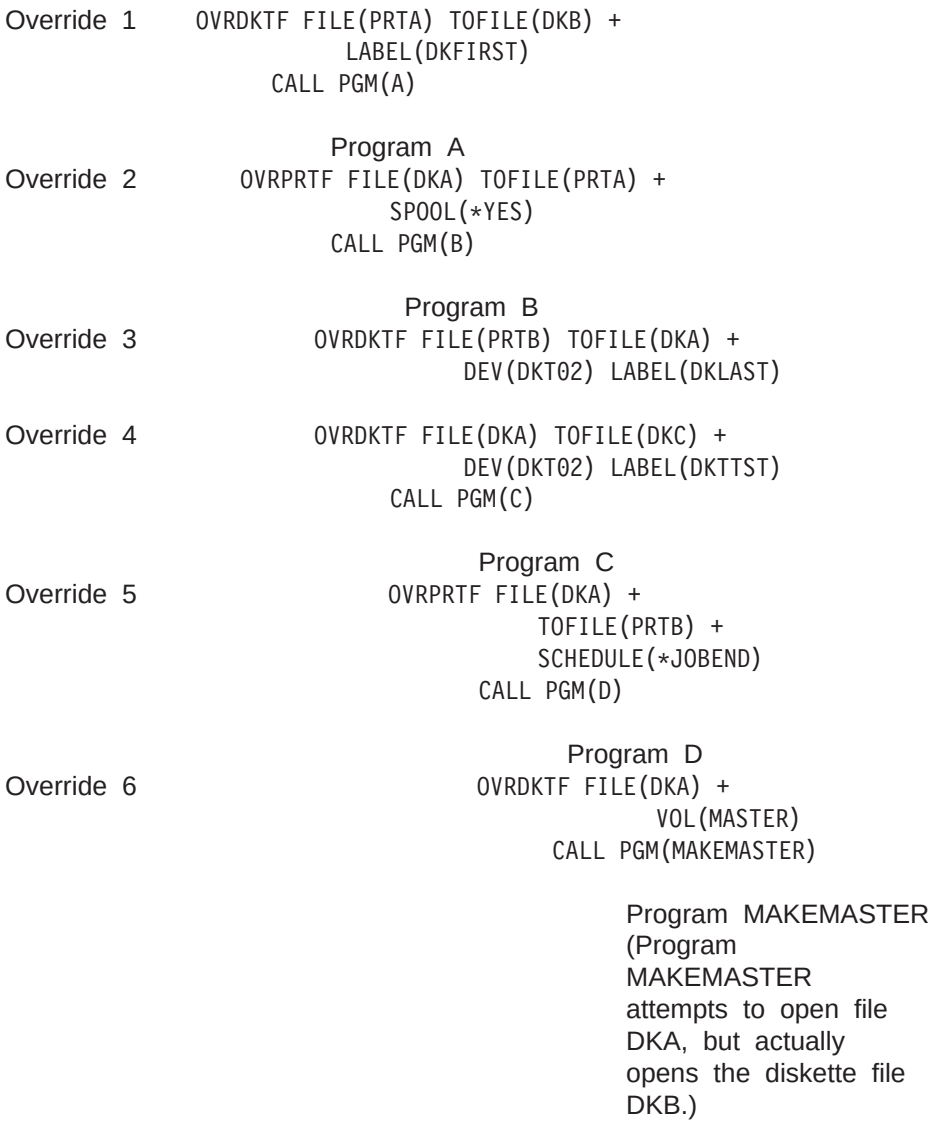
activation group level. The system does not process this override until it has processed all overrides with call levels greater than or equal to the call level of the oldest procedure in activation group 8. In this example, the call level of the oldest procedure in activation group 8 is 3. Therefore, the system will process all call level overrides that are issued at call levels greater than or equal to 3 before processing the activation group override that is issued at call level 8.

4. The system searches call level 7 for applicable overrides that were scoped to the call level. Because the override issued at call level 7 is scoped to the call level, it is processed. The LPI(9) attribute is assigned to file ZZZ.
5. The system searches call level 6 for applicable overrides that were scoped to the call level. Notice that call level 6 is in activation group 21. There is an override for file ZZZ; however, it is scoped to the activation group level of activation group 21. The system ignores this override altogether because it is scoped to an activation group other than activation group 8.
6. The system searches call level 5 for applicable overrides that were scoped to the call level. There are no such overrides. (The override issued in call level 5 is for file YYY and does not apply.)
7. The system searches call level 4 for applicable overrides that were scoped to the call level. Because the override issued at call level 4 is scoped to the call level, it is processed. The CPI(12) attribute is assigned to file ZZZ.
8. The system searches call level 3 for applicable overrides that were scoped to the call level. Because the override issued at call level 3 is scoped to the call level, it is processed. Notice that the file being opened has been changed to YYY from ZZZ. The DEV(P1) attribute is assigned to file YYY. The LPI(9) attribute is changed to LPI(6) and is assigned to file YYY.

   Call level 3 is the call level of the oldest procedure in activation group 8. Therefore, any overrides (for file YYY) that were scoped to the activation group level of activation group 8 are processed. The override issued at call level 9 is processed. This changes the LPI(6) attribute to LPI(5).
9. The system searches call level 2 for applicable overrides that were scoped to the call level. The override issued in call level 2 is processed. This assigns the FOLD(*YES) attribute to file YYY.
10. The system searches call level 1 for applicable overrides that were scoped to the call level. There are no such overrides.
11. The system searches the job level for applicable overrides that were scoped to the job level. Because, the override issued in call level 5 was scoped to the job level and it is for file YYY, it is processed. This changes the DEV(P1) attribute to DEV(P2).

How the system processes overrides-scenario 2

When several overrides that override the file type to be used by a program are applied, only the attributes specified on the overrides of the same type as the final file are applied. In the following example, assume that program MAKEMASTER attempts to open the diskette file DKA:



In the preceding example, the file that program MAKEMASTER actually opens is the diskette file DKB because of the following reasons:

- Override 6, which is applied first, does not cause file DKA to be overridden with any other file.
- Override 5, which is applied second, causes file DKA to be overridden with printer file PRTB.
- Override 4 is ignored at this level because override 5 changed the file name to PRTB.
- Override 3, which is applied third, causes file PRTB to be overridden with diskette file DKA.
- Override 2, which is applied fourth, causes file DKA to be overridden with printer file PRTA.
- Override 1, which is applied last, causes file PRTA to be overridden with diskette file DKB.

Therefore, the file that program MAKEMASTER opens is the diskette file DKB. Because file DKB is a diskette file, the system overrides only those attributes that

are specified on the Override with Diskette File (OVRDKTF) commands: VOL(MASTER) from override 6; DEV(DKT02) from override 3; and LABEL(DKFIRST) from override 1.

The attributes specified on the Override with Printer File (OVRPRTF) commands are ignored (even though they might have been allowed on the OVRDKTF commands). Refer to “Redirecting files” on page 65 for more information on the effect of overrides that change the file type.

Processing overrides: general principles

The system processes overrides according to the following general principles:

- Overrides applied include any that are in effect at the time a file is opened by an application program, when a program that opens a file is compiled, or when certain system commands are used. (See “Overriding file attributes” on page 37, “Applying overrides when compiling a program” on page 54, and “Effect of overrides on some commands” on page 35). Thus, any overrides that are to be applied must be specified either before the file is opened by a program or before a program that opens the file is compiled. It is not necessary that overrides must be supplied for every file that is used in a program. Any file name for which no override is supplied is used as the actual file name.
- Override commands that are scoped to the job level remain in effect until they are replaced, deleted, or until the job in which they are specified ends. For more information on deleting overrides, see “Deleting overrides” on page 55.
- There can be only one active override for a file at each level (job level, activation group level, or call level). If more than one override for the same file exists at the same level, the most recent one is active.

  For an example of how the system processes overrides when more than one override for the same file exists at the same level, see “Overrides to the same file at the same call level: scenario 1” on page 50.
- Override commands that are scoped to the job level apply to all programs that are running in the job regardless of the call level or activation group in which the overrides are specified.
- Override commands that are scoped to an activation group level apply to all programs that are running in the activation group regardless of the call level in which the overrides are specified.
- An override command (scoped to the call level) that is entered interactively exists at the call level for the caller of that command processor. For example, an override (scoped to the call level) that is entered on the command entry display cannot be deleted or replaced from a command processor that is called from the command entry display.
- The call level of an override (scoped to the call level) that is coded in a CL program is the call level of the CL program.
- An override (scoped to the call level) outside a program in a batch job takes the call level of the batch job command processor.
- If an override command (scoped to the call level) is run using a call to the QCMDEXC program, the override takes the call level of the program that called the QCMDEXC program. For an example, see “CL program overrides” on page 51.

- Exits (ENDPGM, RETURN, or abnormal exits) from a call operation delete overrides scoped to that call level. However, they do not delete overrides that are issued in that call level when they are scoped to the activation group level or the job level.

  For an example, see “Effect of exits on overrides: scenario”.

- The TFRCTL command causes one program to be replaced by another program at the same call level. The program, to which control is transferred, runs at the same call level as the program that contained the TFRCTL command. An override command in a program that transfers control to another program is not deleted during the transfer of control.

  For an example, see “Effect of TFRCTL on overrides-Scenario” on page 49.

- Several overrides (possibly one per call level, one at the activation group level, and one at the job level) to a single file are allowed. They are processed according to the priorities in “Processing priority of overrides” on page 43.

  For an example of processing overrides, see “How the system processes overrides-scenario 1” on page 43.

- You can protect an override from being overridden by overrides at lower call levels, the activation group level, and the job level; specify SECURE(*YES) on the override. For an example, see “Securing files against overrides” on page 51.

Effect of exits on overrides: scenario

Exits (ENDPGM, RETURN, or abnormal exits) from a call operation delete overrides that are scoped to that call level. However, they do not delete overrides that are issued in that call level that are scoped to the activation group level or the job level. For example, a RETURN command deletes all overrides scoped to that call level. Thus, overrides that are scoped to the call level in called programs that end with a RETURN or ENDPGM command do not apply to the calling program. This is not true for programs that use the Transfer Control (TFRCTL) command.

In Figure 8 on page 49, the RETURN command deletes the first override in program B, and FILE X is opened in program A. However, the RETURN command does not delete the second override because it is scoped to the job level. FILE B is opened in program A when program A processes the Open FILE A command.

```
                    Program  A
                        .
                        .
                        .

                    CALL PGM(B)

                         Program  B
Override  1      OVRDBF FILE(X) FILE(Y)
Override  2      OVRDBF FILE(A) TOFILE(B) +
                              OVRSCOPE(*JOB)
                            .
                            .
                            .

                        RETURN

                  OPEN FILE X
                       .
                       .
                       .
                  OPEN FILE A
```

Figure 8. Example of Effect of Exits on Overrides

Effect of TFRCTL on overrides-Scenario

The TFRCTL command replaces one program with another program at the same call level. The program to which control is transferred runs at the same call level as the program that contained the TFRCTL command. An override command in a program that transfers control to another program is not deleted during the transfer of control. In the following example, program A transfers control to program B, and program B runs in the same call level as program A. The Override with Database File (OVRDBF) command causes the file to be positioned at the last record of the member when it is opened and is used for both programs A and B.

```
CALL PGM(A)
```

Program A

```
OVRDBF FILE(INPUT) POSITION(*END)
```

(INPUT is opened and positioned at the last record of the member and closed after processing.)

```
TFRCTL PGM(B)
```

Program B

(INPUT is opened and positioned at the last record of the member.)

Overrides to the same file at the same call level: scenario 1

When you enter two overrides for the same file name at the same call level, the second override replaces the first override. This allows you to replace an override at a single call level, without having to delete the first override (see “Deleting overrides” on page 55). For example:

```
Override 1     OVRDKTF FILE(QDKTSRC) LABEL(X)
                       CALL PGM(REORDER)

Override 2     OVRDKTF FILE(QDKTSRC) LABEL(Y)
                       CALL PGM(REORDER)
```

Assume that program REORDER uses the diskette file QDKTSRC. Override 1 causes the first call to program REORDER to use the source file with a label of X for its processing. Override 2 causes the second call to program REORDER to use the source file with a label of Y for its processing.

Overrides to the same file at the same call level: scenario 2

When you enter two overrides for the same file name at the same call level, the second override replaces the first override.

In the following example, when the program attempts to open FILE A, FILE B overrides FILE A because of override 2. Because only one override can be applied for each call level, the system ignores override 1, and the file opened by the program is FILE B.

```
                              Program A
                                 .
                                 .
                                 .
Override 1              OVRDBF FILE(B) TOFILE(C)
Override 2              OVRDBF FILE(A) TOFILE(B)
                                 .
                                 .
                                 .
                            OPEN FILE A
                                 .
                                 .
                                 .
```

To open FILE C, replace the two Override with Database File (OVRDBF) commands with the following command:

```
OVRDBF FILE(A) TOFILE(C)
```

This does not prevent applying an override at the same call level or job level in which the file is created. Regardless of which attribute is encountered first, file attributes on the override take the place of corresponding attributes on the create statement for the file.

CL program overrides

If a CL program overrides a file and then calls a high-level language program, the override remains in effect for the high-level language program. However, if a high-level language program calls a CL program that overrides a file, the system deletes the override automatically when control returns to the high-level language program.

High-level language program:

```
CALL PGM(CLPGM1)
```

```
CL Program CLPGM1
OVRDKTF FILE(DK1) TOFILE(MSTOUT)
   .
   .
   .
ENDPGM
```

High-level language program:

```
OPEN DK1
```

The file opened is DK1, not MSTOUT. This is because the system deletes the override in the CL program when the CL program ends.

To perform an override from a high-level language program, call the QCMDEXC program from the high-level language program. The override specified on the QCMDEXC command, takes the call level of the program that called QCMDEXC.
High-level language program:

```
CALL QCMDEXC PARM('OVRDKTF FILE(DK1) +
  TOFILE(MSTOUT)' 32)
OPEN DK1
```

The file MSTOUT opens because of the override that is requested by the call to the QCMDEXC program.

In an actual program, you might want to use data that is supplied by the program as a parameter of the override. You can do this by using program variables in the call to QCMDEXC. For more information on the use of program variables, refer to the appropriate language information.

Securing files against overrides

On occasion, you may want to prevent the person or program that calls your program from changing the file names or attributes you have specified. You can prevent additional file overrides by coding the SECURE(*YES) parameter on a file override command for each file that needs protection. This protects your file from overrides at lower call levels, the activation group level, and the job level.

The following shows an example of a protected file:

```
Override 1     OVRPRTF FILE(PRINT1) SPOOL(*NO)

Override 2     OVRDBF FILE(NEWEMP) TOFILE(OLDEMP) +
                         MBR(N67)
                 CALL PGM(CHECK)

                         Program CHECK
Override 3         OVRDBF FILE(INPUT) +
                             TOFILE(NEWEMP) MBR(N77) +
                             SECURE(*YES)
                           CALL PGM(EREPORT)

                                 Program EREPORT
                                 (NEWEMP and PRINT1 are opened.)

Override 4         OVRDBF FILE(INPUT) +
                             TOFILE(NEWEMP) MBR(N77)
                       CALL PGM(ELIST)

                               Program ELIST
                               (OLDEMP and PRINT1 are opened.)
```

When the example calls program EREPORT, it attempts to open the files INPUT and PRINT1. EREPORT actually opens file NEWEMP, member N77. Because override 3 specifies SECURE(*YES), the system does not apply override 2. When the example calls program ELIST, it also attempts to open the files INPUT and PRINT1. ELIST actually opens files OLDEMP, member N67. Because override 4 has the same name as override 3 and is at the same call level as override 3, it replaces override 3. Thus, the file is no longer protected from overrides at lower call levels, and the system applies override 2 for program ELIST.

PRINT1 is affected only by override 1, which is in effect for both programs EREPORT and ELIST.

Using a generic override for printer files

The OVRPRTF command allows you to have one override for all the printer files in your job with the same set of values. Without the generic override, you would have to do a separate override for each of the printer files.

Applying OVRPRTF with *PRTF: scenario

You can apply the OVRPRTF command to all printer files by specifying *PRTF as the file name.

The OVRPRTF command with *PRTF is applied if there is no other override for the printer file name at the same call level. The following example shows how *PRTF works:

```
Override 1     OVRPRTF FILE(OUTPUT) COPIES(6) +

                                    LPI(6)
Override 2     OVRPRTF FILE(*PRTF) COPIES(1) +
                                    LPI(8)
                 CALL PGM(X)
```

When program X opens the file OUTPUT, the opened file has the following attributes:

COPIES(6)
From Override 1

LPI(6) From Override 1

When program X opens the file PRTOUT (or any printer file other than OUTPUT), the opened file has the following attributes:

COPIES(1)
From Override 2

LPI(8) From Override 2

Applying OVRPRTF with *PRTF from multiple call levels: scenario

The following example shows how printer-file overrides are applied from multiple call levels by using the *PRTF value.

```
                    Program A
Override 1    OVRPRTF FILE(*PRTF) COPIES(1)
Override 2    OVRPRTF FILE(PRT2) COPIES(2)
Override 3    OVRPRTF FILE(PRT4) COPIES(2)
                   CALL PGM(B)

                    Program B
Override 4      OVRPRTF FILE(*PRTF) LPI(4)
Override 5      OVRPRTF FILE(PRT3) LPI(8)
Override 6      OVRPRTF FILE(PRT4) LPI(8)
                   CALL PGM(X)
```

When program X opens the file PRT1, the opened file has the following attributes:

COPIES(1)
From Override 1

LPI(4) From Override 4

Because no specific overrides are found for PRT1, *PRTF overrides (1 and 4) are applied.

When program X opens the file PRT2, the opened file has the following attributes:

COPIES(2)
From Override 2

LPI(4) From Override 4

Because no specific override is found for PRT2 in program B, override 4 is applied. In program A, override 2 specifies PRT2 and is applied.

When program X opens the file PRT3, the opened file has the following attributes:

COPIES(1)
From Override 1

LPI(8) From Override 5

In program B, override 5 specifies PRT3 and is applied. Because no specific override is found for PRT3 in program A, override 1 is applied.

When program X opens the file PRT4, the opened file has the following attributes:

COPIES(2)
From Override 3

LPI(8) From Override 6

In program B, override 6 specifies PRT4 and is applied. In program A, override 3 specifies PRT4 and is applied.

Applying overrides when compiling a program

Overrides may be applied at the time a program is being compiled for either of two purposes:

- To select the source file
- To provide external data definitions for the compiler to use in defining the record formats to be used on I/O operations

Overrides to the source file are handled just like any other override. They may select another file, another member of a database file, another label for diskette or tape, or change other file attributes.

You can also apply overrides to files that are used within the program being compiled, if they are being used as externally described files in the program. These files are not opened at compile time, and thus the overrides are not applied in the normal manner. These overrides are used at compile time only to determine the file name and library that will be used to define the record formats and fields for the program to use I/O operations. Any other file attributes specified on the override are ignored at compile time. It is necessary that these file overrides be active at compile time only if the file name specified in the source for the program is not the file name that contains the record formats that the application needs.

The file name that is opened when the compiled program is run is determined by the file name that the program source refers to, changed by whatever overrides are in effect at the time the program runs. The file name used at compile time is not kept. The record formats in the file that is actually opened must be compatible with those that were used when the program was compiled. Obviously, the easiest way to assure that records are compatible is to have the same overrides active at run time that were active at compile time. If your program uses externally described data and a different field level file is used at run time, it is usually necessary to specify LVLCHK(*NO) on the override. See “Redirecting files” on page 65 for details.

The following example shows how overrides work when compiling a program:

```
Override 1     OVRDBF FILE(RPGSRC) +
                        TOFILE(SRCPGMS) MBR(INVN42)
Override 2     OVRPRTF FILE(OUTPUT) TOFILE(REPORTS)
               CALL PGM(A)

                          Program A
Override 3               OVRPRTF FILE(LISTOUT) +
                                 TOFILE(OUTPUT)
Override 4               OVRDBF FILE(RPGSRC) WAITFILE(30)
                               CRTRPGPGM PGM(INVENTORY) +
                                    SRCFILE(RPGSRC)
                                RETURN

Override 5     OVRPRTF FILE(LISTOUT) +
                           TOFILE(REPORTS) LPI(8)
               CALL PGM(INVENTORY)
```

The program INVENTORY opens the printer file REPORTS in place of printer file LISTOUT and creates output at 8 lines per inch.

The program INVENTORY is created (compiled) from the member INVN42 in the database file SRCPGMS. Override 4, which is applied first, overrides an optional file attribute. Override 1, which is applied last, causes the file RPGSRC to be overridden with the database file SRCPGMS, member INVN42.

The program INVENTORY is created with the printer formats from the file REPORTS. Assume that the source for the program INVENTORY, which is taken from file SRCPGMS and member INVN42, contains an open to the printer file LISTOUT. Override 3, which is applied first, causes the file LISTOUT to be overridden with OUTPUT. Override 2, which is applied last, overrides OUTPUT with REPORTS. Other attributes may be specified here, but it is not necessary because only the record formats are used at compile time.

At run time, override 3 is no longer active, because program A has ended. Therefore override 2 has no effect on LISTOUT. However, override 5, which is active at run time, replaces LISTOUT with REPORTS and specifies 8 lines per inch. Because the same file is used for compilation and run-time, you can leave level checking on.

Deleting overrides

When a program that has been called returns control to the calling program, any overrides specified in the call level of the called program are deleted. This does not include overrides that are scoped to the activation group level or the job level. Overrides that are scoped to the activation group level remain in effect until they are explicitly deleted, replaced, or until the activation group in which they are specified is deleted. Overrides that are scoped to the job level remain in effect until they are explicitly deleted, replaced, or until the job in which they are specified ends.

When control is transferred to another program (TFRCTL command), the overrides in the call level of the transferring program are not deleted.

To delete overrides:

You use the Delete Override (DLTOVR) command to explicitly delete overrides on your system. The DLTOVR command can delete overrides that are scoped to the call level, activation group level, or the job level. To delete overrides that are scoped to the activation group level, you do not need to specify a value for the OVRSCOPE parameter because OVRSCOPE(*ACTGRPDFN) is the default. To delete overrides that are scoped to the job level, you must specify OVRSCOPE(*JOB) on the DLTOVR command.

To identify an override, use the file name that is specified on the FILE parameter of the override command. You can delete all overrides at the current level (call level, activation group level, or job level) by specifying value *ALL for the FILE parameter.

See the following topics for additional information on deleting overrides:

- "Deleting overrides: scenario 1"
- "Deleting overrides: scenario 2"

Deleting overrides: scenario 1

In the following example, assume that all the commands are entered at the same call level:

```
Override 1                OVRDBF FILE(DBA) +
                                 TOFILE(DBB)
Override 2                OVRPRTF FILE(PRTC) +
                                 COPIES(2)
Override 3                OVRDKTF FILE(DKT) +
                                 EXCHTYPE(*BASIC)
Delete Override 1    DLTOVR FILE(DBA)
Delete Override 2    DLTOVR FILE(*ALL)
```

Delete override 1 causes override 1 to be deleted. Delete override 2 causes the remaining overrides (overrides 2 and 3) to be deleted.

Deleting overrides: scenario 2

In the following example, assume that commands 1, 2, and 14 are entered interactively, at call level 1:

```
                Program A (in user default activation group)
Command 1       OVRDBF FILE(DBA) TOFILE(DBB) SECURE(*YES)
Command 2       CALL PGM(B)

                  Program B (in activation group 4)
Command 3         OVRPRTF FILE(DBB) TOFILE(PRTC) LPI(6) OVRSCOPE(*CALLLVL)
Command 4         OVRDBF FILE(DBC) TOFILE(DBD) OVRSCOPE(*JOB)
Command 5         TFRCTL PGM(C)

                  Program C
Command 6         OVRDKTF FILE(DKTE) TOFILE(DKTF) OVRSCOPE(*CALLLVL)
Command 7         CALL PGM(QCMDEXC) +
                    PARM('OVRDSPF FILE(DSPG) TOFILE(DSPH)' 31)
Command 8         DLTOVR FILE(DBA DBB) LVL(*)
Command 9         MONMSG MSGID(CPF9841)
Command 10        CALL PGM(QCMDEXC) PARM('DLTOVR FILE(*ALL) LVL(*)' 24)

Command 11        DLTOVR FILE(DBC) OVRSCOPE(*JOB)
Command 12        DLTOVR FILE(DSPG)
Command 13        RETURN
Command 14      DLTOVR FILE(*ALL)
```

Command 1 causes an override at level 1 from file DBA to file DBB.

Command 2 calls program A and creates a new call level (call level 2).

Command 3 causes an override at level 2 from file DBB to file PRTC. Also, the LPI attribute of file PRTC is overridden to 6.

Command 4 causes an override at the job level from file DBC to file DBD.

Command 5 transfers control from program A to program B at the same call level (call level 2).

Command 6 causes an override at level 2 from file DKTE to file DKTF.

Command 7 causes an override at activation group level 4 from file DSPG to file DSPH. OVRSCOPE(*ACTGRPDFN) is the default.

Command 8 deletes any overrides of files DBA and DBB at level 2. The override specified by command 3 is deleted, but the override specified by command 1 is not deleted. Because an override for DBA cannot be found at level 2, the override-not-found escape message (CPF9841) is sent.

Command 9 monitors for a message to prevent a function check, but it specifies no action to be taken if the message is sent.

Command 10 deletes all remaining overrides at level 2. The override specified by command 6 is deleted, but the overrides specified by commands 1, 4, and 7 are not deleted.

Command 11 deletes overrides to file DBC that are scoped to the job level. The override specified by command 4 is deleted.

Command 12 deletes the override to file DSPG that was scoped to activation group level 4 by command 7.

Command 13 causes a return to level 1, and level 2 is deleted. If any overrides were specified at level 2 (scoped to the call level) between command 10 and

command 12, they are deleted at this point. Also, if any overrides were specified at level 2 (scoped to the activation group level) between command 10 and 12, they are deleted assuming that activation group 4 is ended after the RETURN.

Command 14 causes all overrides specified at call level 1 to be deleted. The override specified by command 1 is deleted.

Note: Command 14 would not delete any overrides that were scoped to the job level. (However, there are none in this example at the time command 14 is issued). In general, to delete all overrides at the job level, you would have to specify DLTOVR FILE(*ALL) OVRSCOPE(*JOB).

Displaying overrides

You can use the Display Override (DSPOVR) command to display file overrides at the job level, the activation group level, and at multiple call levels for a job. You can display all file overrides or overrides for a specific file.

The file overrides may be merged before being displayed. A merged override is the result of combining overrides from the job level to the current level or any specified call level, producing a composite override which will be applied when the file is used at the specific call level. The current call level is the call level of the program that is currently running. This program is the last program name that is displayed on the call stack. This command may be requested from either a batch or interactive environment. You can also access this function from option 15 (Display file overrides) from the Work with Job menu (using the WRKJOB command) or by selecting option 15 (Display file overrides) from the Display Job menu (using the DSPJOB command).

For additional information:

See the following topics for more information on displaying overrides:

- “Displaying all overrides for a specific activation group: scenario”
- “Displaying merged file overrides for one file: scenario” on page 59
- “Displaying all file overrides for one file: scenario” on page 59
- “Displaying merged file overrides for all files: scenario” on page 59
- “Displaying overrides with WRKJOB: scenario” on page 59
- “Displaying overrides: comprehensive scenario” on page 60
- “Displaying overrides: tips” on page 64

Displaying all overrides for a specific activation group: scenario

To display all overrides for a specific activation group, you type:

```
DSPOVR FILE(REPORTS) ACTGRP(*)
```

This displays all the overrides for the REPORTS file for the activation group in which the override is issued. ACTGRP(*) is the default and is shown here for illustration purposes. To specify an activation group other than the one the command is to be issued in, specify the name of the activation group on the ACTGRP parameter.

Displaying merged file overrides for one file: scenario

To display the merged file override for a particular file at a specific call level, you type:

```
DSPOVR FILE(REPORTS) MRGOVR(*YES) LVL(3)
```

This command produces a display that shows the merged override for the file REPORTS at call level 3 with text descriptions of each keyword and parameter. Any applicable overrides at the job level, the activation group level, and at call levels 1, 2, and 3 are used to form the merged override, but overrides at higher call levels are ignored. If the call level specified is not active, all applicable overrides up to the current level are used.

Displaying all file overrides for one file: scenario

To display all file overrides for a specific file up to a specific call level, you type:

```
DSPOVR FILE(REPORTS) MRGOVR(*NO) LVL(2)
```

This command produces a display that shows the file name, the call level for which the override was requested, the type of override, and the override parameters in keyword-parameter form. If no file overrides are found for the file up to and including the specified call level, escape message `CPF9842` is sent. If you are using DSPOVR in a CL program, you might want to add a MONMSG command following the DSPOVR command to prevent your program from ending if there are no overrides for the file. This technique is illustrated in some of the examples later in this chapter. For more information on the MONMSG command, refer to the *CL Programming* book.

Displaying merged file overrides for all files: scenario

To display the merged file overrides for all files at the current call level, you type:

```
DSPOVR FILE(*ALL) MRGOVR(*YES) LVL(*)
```

This command produces a display showing the file name, the type of override, and the merged overrides in keyword-parameter form, where only the keywords and parameters entered on the commands are displayed. This is the same as what happens when you type DSPOVR with no parameters. Only those keywords for which parameters were specified are displayed. The associated text descriptions are not displayed. Overrides at call levels greater than 999 are not displayed.

Displaying overrides with WRKJOB: scenario

When overrides are displayed not by the DSPOVR command, but through an option on one of the system interfaces to work with jobs (for example, WRKJOB), all file overrides from the job level to the current call level are displayed. This would be the same as typing the following command:

```
DSPOVR FILE(*ALL) MRGOVR(*NO) LVL(*)
```

This produces a display showing the file name, the level (call level, activation group level, or job level) for which the override was requested, the type of override, and the override parameters in keyword-parameter form for each override.

Because the display overrides function uses a copy of the internal control blocks, overrides that were deleted between the time the display overrides function was

called and the time the output was produced may not be reflected in the output. This can occur only when the overrides in another job are being displayed.

Displaying overrides: comprehensive scenario

The following example is intended only to illustrate what the various forms of the display override command can do. The DSPOVR command is typically entered interactively or added temporarily to a CL program, or to any high-level language program via QCMDEXC, to verify that the proper overrides are in effect at the time a program is called or a file is opened. Assume that commands 1, 2, 3, and 18 are entered at call level 1:

```
                Program A (in the user default activation group)
Command 1       OVRPRTF FILE(PRTA) COPIES(3)
Command 2       OVRDBF FILE(DBC) WAITFILE(*IMMED)
Command 3       CALL PGM(B)

                  Program B (in activation group 5)
Command 4         OVRPRTF FILE(PRTB) TOFILE(PRTA) COPIES(6) +
                    OVRSCOPE(*CALLLVL)
Command 5         OVRDBF FILE(DBC) WAITFILE(60) OVRSCOPE(*CALLLVL)
Command 6         OVRDBF FILE(DBE) TOFILE(DBF) OVRSCOPE(*JOB)
Command 7         DSPOVR FILE(PRTB) MRGOVR(*YES)
Command 8         CALL PGM(C)

                    Program C (in activation group 5)
Command 9           CALL PGM(QCMDEXC) PARM('OVRDSPF FILE(DSPE) +
                      TOFILE(DSPF) OVRSCOPE(*CALLLVL)' 50)
Command 10          OVRDBF FILE(DBC) TOFILE(DBD) OVRSCOPE(*CALLLVL)
Command 11          DSPOVR FILE(DBC) MRGOVR(*NO) LVL(3)
Command 12          DSPOVR FILE(DBD) MRGOVR(*NO) LVL(2)
Command 13          MONMSG MSGID(CPF9842)
Command 14          OVRDSPF FILE(CREDITS) TOFILE(DEBITS)
Command 15          CALL PGM(QCMDEXC) PARM('DSPOVR FILE(*ALL) MRGOVR(*YES) +
                      LVL(*) OUTPUT(*)' 47)
Command 16          RETURN

Command 17        DSPOVR FILE(*ALL) MRGOVR(*NO)
Command 18        RETURN
Command 19      DSPOVR FILE(*ALL) MRGOVR(*NO) LVL(2) OUTPUT(*)
```

Command 1 overrides the value of the COPIES attribute of file PRTA at level 1 to 3.

Command 2 overrides the value of the WAITFILE attribute of file DBC at level 1 to *IMMED.

Command 3 calls program A and creates a new call level, 2.

Command 4 causes an override at level 2 from file PRTB to file PRTA. Also, the command overrides the value of the COPIES attribute to 6.

Command 5 overrides the the value of the WAITFILE attribute for file DBC at level 2 to 60.

Command 6 causes an override of file DBE to file DBF and scopes the override to the job level.

Command 7 displays a merged override for file PRTB at level 2 with text descriptions of each keyword and parameter, as shown in Figure 9 on page 61. The to-file is

PRTA because of command 4, and the COPIES attribute is 3 because of command 1.

```
                    Display Override with Printer File

File  . . . . . :   PRTB
Call level  . . . . . :   *
Merged  . . . . . :   *YES

                                    Keyword    Value
Name of file being overridden . . :   FILE       PRTB
Overriding to printer file  . . . :   TOFILE     PRTA
Library . . . . . :                   *LIBL
Number of copies  . . . . . :   COPIES     3









Press Enter to continue.

F3=Exit   F12=Cancel
```

Figure 9. Override with Printer File Display

Command 8 calls program B and creates the new call level 3.

Command 9 causes an override at level 3 from file DSPE to file DSPF. An override done via a call to the QCMDEXC program takes the call level of the program that called the QCMDEXC program.

Command 10 causes an override of file DBC to file DBD.

Command 11 displays all overrides for file DBC from the job level to level 3, as shown in Figure 10 on page 62. The overrides specified by commands 10, 5, and 2 are displayed in keyword-parameter form. Observe that this form of the DSPOVR command shows all the overrides for the selected file, regardless of redirection. The three overrides that are shown would not be merged because of the name change at level 3.

```
                    Display All File Overrides

Call level  . . . . . :   3

Type options, press Enter.
  5=Display override details

Opt  File    Level  Type  Keyword Specifications
 _   DBC     3      DB    TOFILE(*LIBL/DBD)
 _           2      DB    WAITFILE(60)
 _           1      DB    WAITFILE(*IMMED)









F3=Exit   F5=Refresh   F12=Cancel
```

Figure 10. All File Overrides Display (One File)

Command 12 attempts to display all file overrides for file DBD from the job level to level 2. Because no overrides for file DBD exist at levels 1 or 2, no overrides are displayed, and the override-not-found escape message (CPF9842) is sent.

Command 13 monitors for message CPF9842 on the preceding command. The monitor specifies no action to be taken, but will prevent a function check if the message is sent.

Command 14 causes an override of the display file CREDITS to the display file DEBITS. The override is scoped to the activation group level of activation group 5. OVRSCOPE(*ACTGRPDFN) is the default.

Command 15 displays the merged overrides at the job level to call level 3 for all files in keyword-parameter form, as shown in Figure 11 on page 63. File DBC is overridden to file DBD because of command 10 (commands 5 and 2 are therefore not effective). File DSPE is overridden to file DSPF because of command 9. File PRTB is overridden to file PRTA and COPIES(3) because of commands 4 and 1. File DBE is overridden to file DBF because of command 6. The file DEBITS overrides the file CREDITS because of command 14.

```
                Display All Merged File Overrides

Call level  . . . . . :   *

Type options, press Enter.
  5=Display override details  8=Display contributing file overrides


Opt  File     Type  Keyword Specifications
 _   DSPE     DSP   TOFILE(*LIBL/DSPF)
 8   PRTB     PRT   TOFILE(*LIBL/PRTA) COPIES(3)
 _   DBC      DB    TOFILE(*LIBL/DBD)
 _   PRTA     PRT   COPIES(3)
 _   DBE      DB    TOFILE(*LIBL/DBF)
 _   CREDITS  DSPF  TOFILE(*LIBL/DEBITS)









F3=Exit   F5=Refresh   F11=All file overrides   F12=Cancel
```

Figure 11. All Merged File Overrides Display

If you enter a 5 on the line for PRTB, you get a detail display like the one shown in Figure 9 on page 61. If you enter an 8 on this same line, you get a display showing commands 4 and 1 on separate lines, as shown in Figure 12. These are the overrides that were merged to form the PRTB override.

```
                Display Contributing File Overrides

File  . . . . . :   PRTB
Call level  . . . . . :   *

Type options, press Enter.
  5=Display override details

Opt  Level  Type  Keyword Specifications
 _     2    PRT   TOFILE(*LIBL/PRTA) COPIES(6)
 _     1    PRT   COPIES(3)









F3=Exit   F5=Refresh   F12=Cancel   F14=Display previous override
```

Figure 12. Contributing File Overrides Display

Command 16 causes a return to level 2, and level 3 is deleted. The overrides issued at level 3 that are scoped to the call level are implicitly deleted. The override issued by command 14 is not deleted because it is scoped to the activation group level.

Command 17 displays all overrides issued for the job level to the current call level (level 2), as shown in Figure 13. The overrides specified in commands 1, 2, 4, 5, 6, and 14 display in keyword-parameter form. The override issued in command 10 is not displayed because call level 3 is no longer active. Pressing F11 on this display allows you to see a display that is similar to the one shown in Figure 11 on page 63.

```
                     Display All File Overrides

Call level  . . . . . . . . . . . :   *

Type options, press Enter.
  5=Display override details

Opt  File       Level  Type  Keyword Specifications
 _   CREDITS  *ACTGRP  PRT   TOFILE(*LIBL/DEBITS)
 _   PRTB           2  PRT   TOFILE(*LIBL/PRTA) COPIES(6)
 _   DBC            2  DB    WAITFILE(60)
 _                  1  DB    WAITFILE(*IMMED)
 _   PRTA           1  PRT   COPIES(3)
 _   DBE         *JOB  DB    TOFILE(*LIBL/DBF)









F3=Exit   F5=Refresh   F11=All merged file overrides   F12=Cancel
```

Figure 13. All File Overrides Display (All Files)

Command 18 causes a return to level 1, and level 2 is deleted. The overrides issued at level 2 that are scoped to the call level are implicitly deleted. The override that is caused by command 14 (scoped to the activation group level) is implicitly deleted when activation group 5 ends. In this example, assume that activation group 5 is a nonpersistent activation group and that ends when command 18 processes. The override caused by command 6 is not deleted.

Command 19 displays all overrides for the job level to call level 2 in keyword-parameter form. Because level 2 is no longer active, only the overrides scoped to the job level (command 6) and those specified at level 1 in commands 1 and 2 are displayed.

Displaying overrides: tips

Note that when specifying a call level, as in the first two examples in this section, the call level on which you first entered override commands may not be level 1. Depending on the contents of the first program and first menu specified in your user profile, and any other programs or menus you may have come through, you may have entered your first override commands at level 3 or 4. You may enter WRKJOB and select option 11 (call stack) to see what programs are running at lower call levels.

Unless you know exactly what you want to see, it is usually best to request the override display with no parameters, because options on the basic override display allow you to select a detailed display of any override you are interested in. The specific options available are:

- From the merged display of all overrides, you can request the display that is not merged, as in “Displaying overrides with WRKJOB: scenario” on page 59.
- From the unmerged display of all overrides, you can request the merged display.
- From the merged display of all overrides, you can request a merged detail display of any override, equivalent to the command in “Displaying merged file overrides for one file: scenario” on page 59.
- From the merged display of all overrides, you can request a display of all the individual overrides that contributed to the merged display, showing the level (call level or job level) for which each was requested.
- From either the display of contributing overrides or the display (not merged) of all overrides, you can request a detail display of the override for a particular file at a single call level.

Redirecting files

File redirection lets you use overrides to direct data input or output to a device of a different type; for example, to send data that was intended for a diskette to a printer instead. This use of overrides requires somewhat more foresight than the override applications listed above, because the program must be able to accommodate the different characteristics of the two devices involved.

To override to a different type of file, use the override command for the new type of file. For example, if you are overriding a diskette file with a printer file, use the Override with Printer File (OVRPRTF) command.

This section applies to using an application program only. System code may or may not support file redirection. Refer to “Effect of overrides on some commands” on page 35 for rules on how system code processes overrides.

You use the OVRDBF command to redirect a file to a Distributed Data Management (DDM) file. If the remote system is another AS/400 system, all normal rules discussed in this chapter apply. If the remote system is not an AS/400 system or System/38, then normally you should not specify an expiration date or end-of-file delay. For more information, refer to the *Distributed Data Management* book.

When you replace the file that is used in a program with another file of the same type, the new file is processed in the same manner as the original file. If you redirect a field-level file, or any other file that contains externally described data, you should usually specify LVLCHK(*NO) or recompile the program. Even when you turn level checking off, the record formats in the file must remain compatible with the records in the program. If the formats are not compatible, the results cannot be predicted.

Overrides that have a TOFILE parameter value other than *FILE remove any database member specifications that may be on overrides applied at higher call levels. The member name will default to *FIRST unless it is specified with the change to the file name or library or on another override at a lower call level.

If you change to a different type of file, the system ignores device-dependent characteristics and records that the system reads or writes sequentially. You must

specify some device parameters in the new device file or the override. The system uses defaults for others. The effect of specific redirection combinations is described later in this section.

The system ignores any attributes that are specified on overrides of a different file type than the final file type. The parameters SPOOL, SHARE, and SECURE are exceptions to this rule. The system accepts the parameters from any override that is applied to the file, regardless of device type.

Planning for redirecting files

Table 8 summarizes valid file redirections.

To use this chart, identify the file type that you want to override in the FROM-FILE columns, and the file type that you want to override in the TO-FILE column. The intersection specifies an I or O or both; this means that the substitution is valid for these two file types when used as input files or as output files.

For instance, you can override a diskette output file with a tape output file, and a diskette input file with a tape input file. The chart refers to file type substitutions only. That is, you cannot change the program function by overriding an input file with an output file.

Table 8. File Redirections

	From-File					
To-File	**Printer**	**intersystem communications function (ICF)**	**Dis- kette**	**Display**	**Data- base**	**Tape**
Printer	O*	O	O	O	O	O
ICF	O	I/O O I	O I	I/O O I	O I	O I
Diskette	O	O I	O I	O I	O I	O I
Display	O	I/O O I	O I	I/O O I	O I	O I
Database	O	O I	O I	O I	O I	O I
Tape	O	O I	O I	O I	O I	O I

:

- I=input file O=output file I/O=input/output file
- *=redirection to a different type of printer

Redirecting files: tips

Some redirection combinations present special problems due to the specific characteristics of the device. In particular:

- You should not redirect save files.
- You can redirect nonsequentially processed database files only to another database file or a DDM file.
- You can redirect Display files and ICF files that use multiple devices (MAXDEV or MAXPGMDEV > 1) only to a display file or ICF file.
- Redirecting a display file to any other file type, or another file type to a display file, requires that the program be recompiled with the override active if there are

any input-only or output-only fields. This is necessary because the display file omits these fields from the record buffer in which it does not use them, but other file types do not.

Default actions for redirected files

The charts in this section describe the specific defaults that the system takes when it redirects files, and which defaults it ignores for each redirection combination.

From Printer

To ICF: Records are written to the file one at a time. Printer control information is ignored.

Display: Records are written to the display with each record overlaying the previous record. For program-described files, you can request each record using the Enter key. Printer control information is ignored.

Database: Records are written to the database in sequential order. Printer control information is ignored.

Diskette: The amount of data written on diskette is dependent on the exchange type of the diskette. Diskette label information must be provided in the diskette file or on an override command. Printer control information is ignored. Refer to the *Tape and Diskette Device Programming* book for a description of exchange types.

Tape: Records are written to the tape in sequential order. Tape label information must be specified in the tape file or on an override command. Printer control information is ignored.

From ICF input

To Display: Records are retrieved from the display one at a time. Type in the data for each record and press the Enter key when the record is complete.

Database: Records are retrieved from the database.

Diskette: Records are retrieved in sequential order. Diskette label information must be provided in the diskette file or on an override command. Refer to the *Tape and Diskette Device Programming* book for a description of exchange types.

Tape: Records are retrieved in sequential order. Tape label information must be specified in the tape file or on the override command.

From ICF output

To Printer: Records are printed and folding or truncating is performed as specified in the printer file.

Display: Records are written to the display with each record overlaying the previous record.

Database: Records are written to the database in sequential order.

Diskette: The amount of data written on diskette is dependent on the exchange type of the diskette. Diskette label information must be provided in the diskette file or on an override command. Refer to the *Tape and Diskette Device Programming* book for a description of exchange types.

Tape: Records are written to the tape in sequential order. Tape label information must be specified in the tape file or on the override command.

From ICF input/output

To Display: Input records are retrieved from the display one at a time. Type in the data for each record and press the Enter key when the record is complete. Output records are written to the display with each record overlaying the previous input or output record. Input and output records are essentially independent of each other and may be combined in any manner.

From Diskette input

To ICF: Records are retrieved from the ICF file one at a time.

Display: Records are retrieved from the display one at a time. Type in the data for each record and press the Enter key when the record is complete. A nonfield-level device file must be specified. Diskette label information is ignored.

Database: Records are retrieved in sequential order. Diskette label information is ignored.

Tape: Records are retrieved in sequential order. If a label value is specified in the program, that value is used as the label for the tape file.

From Diskette output

To ICF: Records are written to the ICF file one at a time.

Database: Records are written to the database in sequential order.

Display: Records are written to the display with each record overlaying the previous record. You can request each output record using the Enter key.

Printer: Records are printed and folding or truncating is performed as specified in the printer file.

Tape: Records are written on tape in sequential order.

From Display input

To ICF: Records are retrieved from the ICF file one at a time.

Diskette: Records are retrieved in sequential order. Diskette label information must be provided in the diskette file or on an override command. Refer to the *Tape and Diskette Device Programming* book for a description of exchange types.

Database: Input records are retrieved.

Tape: Records are retrieved in sequential order. Tape label information must be specified in the tape file or on an override command.

From Display output

To ICF: Records are written to the ICF file one at a time.

Database: Records are written to the database in sequential order.

Diskette: The amount of data written on diskette is dependent on the exchange type of the diskette. Diskette label information must be provided in the diskette file or on an override command. Refer to the *Tape and Diskette Device Programming* book for a description of exchange types.

Tape: Records are written on tape in sequential order. Tape label information must be specified in the tape file or on an override command.

Printer: Records are printed and folding or truncating is performed as specified in the printer file.

From Display input/output

To ICF: Input records are retrieved from the ICF file one at a time. Output records are written to the ICF file one at a time. The relationship between the input and output records is determined by the application program.

From Database input (sequentially processed)

To ICF: Records are retrieved from the ICF file one at a time.

Display: Records are retrieved from the display one at a time. Type in the data for each record and press the Enter key when the record is complete. A nonfield-level device file must be specified.

Diskette: Records are retrieved in sequential order. Diskette label information must be provided in the diskette file or on an override command. Refer to the *Tape and Diskette Device Programming* book for a description of exchange types.

Tape: Records are retrieved from tape in sequential order. Tape label information must be specified in the tape file or on an override command.

From Database output (sequentially processed)

To Printer: The number of characters printed is determined by the page size specified. If folding is specified, all of a record is printed.

ICF: Records are written to the ICF file one at a time.

Display: Records are written to the display with each record overlaying the previous record. You can request each output record using the Enter key.

Diskette: The amount of data written on diskette depends on the exchange type of the diskette. Diskette label information must be provided in the diskette file or on an override command. Refer to the *Tape and Diskette Device Programming* book for a description of exchange types.

Tape: Records are written on tape in sequential order. Tape label information must be specified in the tape file or on an override command.

From Tape input

To ICF: Records are retrieved from the ICF file one at a time.

Display: Records are retrieved from the display one at a time. Type in the data for each record and press the Enter key when the record is complete. A nonfield-level device file must be specified. Tape label information is ignored.

Database: Records are retrieved in sequential order. One record is read as a single field. Tape label information is ignored.

Diskette: Records are retrieved in sequential order. If a label value is specified in the program, that value is used as the label for the diskette file.

From Tape output

To Printer: Records are printed, and folding or truncating is performed as specified in the printer file.

ICF: Records are written to the ICF file one at a time. Tape label information is ignored.

Diskette: The amount of data written on diskette depends on the exchange type of the diskette. If a label value is specified in the program, that value is used as the label for the diskette file. Refer to the *Tape and Diskette Device Programming* book for a description of exchange types.

Display: Records are written to the display with each record overlaying the previous record. You can request each output record using the Enter key.

Database: Records are written to the database in sequential order.

Chapter 4. Copying files

You can use the copy function to move data between device files, database files, or both device and database files with the AS/400 field-level sensitive copy function. This function allows you to rearrange, enlarge, or drop any of the fields. You can also define database files.

Copying files

To copy a physical or logical file (the *from-file*) on AS/400 into another physical file (the *to-file*), which does not yet exist, you can use the CPYF command, as in the following example:

```
CPYF   FROMFILE(PERSONNEL/PAYROLL)
   TOFILE(TESTPAY/PAYROLL)  MBROPT(*ADD)
   CRTFILE(*YES)  ERRLVL(10)
```

Full service copy support:

A variety of copy commands that are modified by numerous parameters gives you a great deal of flexibility in the way you copy your data. For instance, you usually can copy your data into existing files (or to-files). As shown in the example above, you can use the CRTFILE parameter on the CPYF or CPYFRMQRYF commands to create the to-file during the copy operation. See “Create the To-File (CRTFILE Parameter)” on page 72 for details.

Copy only the information you need:

The copy function lets you specify selected records and members of your files:

- “Add, replace, and update records (MBROPT parameter)” on page 74
- “Select members to copy” on page 75
- “Select the records to copy” on page 78

Copy across different formats and systems:

- “Copying between different database record formats (FMTOPT parameter)” on page 93. You can copy from a source file to a data file or from a data file to a source file. If the from-file or to-file is a device file, this function is automatic. If both files are database files, you must specify FMTOPT(CVTSRC). If either file is a device file or inline data file, the FMTOPT parameter does not apply.
- “Copy between different systems” on page 122. This is especially important for when you are using Data Warehousing, and when you want to use existing export products from other platforms to move data to the AS/400.

Make the copy function work for your particular needs:

You can accomplish a wide variety of tasks with careful use of the options that are available to you through the copy function.

- “Print records (PRINT, OUTFMT, and TOFILE(*PRINT) parameters)” on page 91
- “Add or change source file sequence number and date fields (SRCOPT and SRCSEQ Parameters)” on page 103
- “Prevent errors when copying files” on page 104

- “Improve copy performance” on page 111
- “Year 2000 support: date, time, and timestamp considerations” on page 112

Create the To-File (CRTFILE Parameter)

To copy a physical or logical file when no to-file exists to receive the data, you can create the to-file by specifying CRTFILE(*YES). Specify the name of the new to-file on the TOFILE parameter. Qualify the name with the name of an existing library for which you have the required authority. (You must also have authority to the CRTPF command). You cannot override the created to-file that you specified to a different file or library.

CRTFILE(*YES) automatically adds members and records to the new file.

The newly created file has certain authorities, capabilities, and a user profile associated with it. For more information, see “Authorities, user profiles, and file capabilities of the created to-file” on page 73. Your system specifies different identifiers and attributes to the new file based on whether you use the CPYF or CPYFRMQRYF command. See “Specifying CRTFILE(*YES) on either the CPYF or CPYFRMQRYF command”.

Specifying CRTFILE(*YES) on either the CPYF or CPYFRMQRYF command

If you specify CRTFILE(*YES) on the CPYF command, the to-file that is created has the same record format and type of access path as the from-file. The file level and the format level identifiers of the new to-file are identical to the file level and the format level identifiers of the from-file. The text of from-file members that are copied is used as the text of any to-file members that are created.

When the from-file is a logical file, the system assigns the following physical file attributes: SIZE(*NOMAX), ALLOCATE(*NO), and CONTIG(*NO). If the from-file is a logical file with multiple record formats, the to-file is created with the format that is specified on the RCDFMT parameter on the CPYF command. See “Select records using a specified record format name (RCDFMT Parameter)” on page 79 for more information on the RCDFMT parameter.

If you specify CRTFILE(*YES) on the CPYFRMQRYF command, the file level and the format level identifiers of the new to-file are generated at the time the new to-file is created. Furthermore, the physical file’s attributes match the first file that is specified on the FILE parameter of the corresponding Open Query File (OPNQRYF) command. However, the system assigns some of the attributes. The file is created with CONTIG(*NO), SIZE(*NOMAX), ALLOCATE(*NO), AUT(*NORMAL) and FILETYPE(*DATA).

The name, type, length, null capability, date, or time format, separators, and decimal positions attributes of each field on the format that is specified are used. The file is created without key fields and is an arrival sequence physical file.

In some cases, the OPNQRYF command changes the format of the format that is specified on the new to-file. The new to-file format may become null-capable when the OPNQRYF command uses one of the following grouping functions:

- %STRDEV
- %VAR
- %SUM
- %AVG
- %MIN
- %MAX

Note: A new to-file with a changed format has a format level identifier that is different from the format level identifier that is specified on the OPNQRYF command.

Authorities, user profiles, and file capabilities of the created to-file

When the Copy File (CPYF) command creates the local physical file, the from-file gives the created to-file all the **authorities** of the from-file. These authorities include public, private, and authorization lists. When CPYFRMQRYF creates the local physical file, the authorities given are of the first file that is specified on the FILE parameter of the corresponding Open Query File (OPNQRYF) command. The authorities include public, private, and authorization lists.

In both cases, the owner of the created to-file is the **user profile** running the copy command. The user running the copy command inherits *ALL authority to the object. This is true unless the user is a member of a group profile and has OWNER(*GRPPRF) specified for the profile.

If you specify OWNER(*GRPPRF), the group profile becomes the owner of the to-file. In this case, if the user profile running the copy command does not have authority to add a member or write data to the new file, the copy command fails.

The created to-file does not maintain the file capabilities of the from-file. The to-file allows update, delete, read, and write operations, regardless of whether the from-file allowed these operations. Following are special considerations for the new to-file:

- If the number of records copied into a member is greater than the maximum size of the created to-file, the to-file is extended without intervention by the system operator.
- If the from-file is an SQL table, view, or index, the created to-file will be a physical file that is not an SQL table. However, when the from-file contains LOBs, datalinks, or user-defined types, the created to-file is an SQL table.
- If the from-file has a trigger program associated with it, the CPYF and CPYFRMQRYF commands do not copy the trigger information to the to-file when the CRTFILE parameter is used.
- If you create a new file (CRTFILE(*YES)) from a file with constraints, the constraint definitions do not copy to the new file.
- If you create a new file (CRTFILE(*YES)) from a file with user-defined functions, the user-defined functions do not copy to the new file.

Add, replace, and update records (MBROPT parameter)

On the CPYF, CPYFRMDKT, CPYFRMQRYF, CPYFRMTAP, or CPYSRCF commands, you can add or replace existing data in the to-file by specifying different attributes on the MBROPT parameter. The CPYF command also allows you to update duplicate key records and add non-duplicate key records to a to-file member. You can do these tasks by specifying *REPLACE, specifying *ADD, or specifying *UPDATE on the MBROPT parameter (see page Specifying *REPLACE).

Specifying *REPLACE

By specifying *REPLACE, you essentially clear the member. The copied records are the only records in the member when the operation completes. You must have authority to clear the member in order to specify MBROPT(*REPLACE).

For copy commands except the CPYFRMQRYF command, when you specify *REPLACE, copy command processing fails if the from-file does not contain any records. When you specify *REPLACE on the CPYFRMQRYF command, the to-file member will be cleared even if the open query file contains no records.

*REPLACE is the default value for the CPYSRCF command. All other copy commands have the default value of *NONE; however, *NONE is valid only for copying to a device file.

Specifying *ADD

By specifying *ADD, each record copied is added to the end of the existing records in the member. This is always true, even for keyed files. (With keyed files, the added records appear merged in key sequence when accessed through keyed access path.) When you specify *ADD, the copy completes normally even if the from-file contains no records.

Specifying *UPDADD

When you specify *UPDADD on the CPYF command, a from-file key value builds before the from-file record moves into the to-file. The from-file builds this key value by using the key specifications of the to-file. Before the key value is built, the system performs any necessary field or data mapping, data conversion, or record selection. The system checks the to-file to see if this key value already exists in it (duplicate key of the from-file data). If the key value does exist in the to-file,the from-file record that contains the key value updates that to-file record.

The following apply if you specify MBROPT(*UPDADD) on the CPYF command:

- The to-file must be a local database physical file that contains a primary or unique key.
- You may not specify CRTFILE(*YES). The to-file must exist before you run CPYF.
- CPYF cannot copy from multiple formats.
- Detected duplicate keys are not skipped but updated with the new from-file record value. Duplicate key errors (CPF5026) are not included as ERRLVL errors.
- CPF5027 will be included as an ERRLVL error. This error can occur if another process has a record that is locked. To avoid this error, you may want to

pre-allocate the to-file within your job before performing the CPYF. You can use the WAITRCD parameter on the CRTPF and CHGPF commands to limit the length of time spent waiting for a record lock to be released in the to-file.

- All existing FMTOPT values are allowed. However, when using MBROPT(*UPDADD), take care to avoid updating records that you do not want to update. Also avoid updating the same record multiple times when it is not desired.
- Nulls are not used in determining duplicate key values if FMTOPT(*NOCHK) is specified or if the from-file is a device file.
- You must have the minimum following authorities to the to-file:
  - Object operational (*OBJOPR)
  - Add (*ADD)
  - Update (*UPD)

Select members to copy

AS/400 gives you several options for copying file members:

- “Copy all members or labels within a file” on page 76
- “Copy only certain members or labels within a file” on page 76

“Copying file members: overview” gives an explanation of how the system handles this process.

For more information:

For more details, see the following topics:

- “Specifying the label identifier or member name for the copy operation” on page 77
- “Special considerations for the Override Database File (OVRDBF), Override Diskette File (OVRDKTF), and Override Tape File (OVRTAPF) commands” on page 77

Copying file members: overview

You can copy multiple database members or diskette labels to corresponding like-named to-file members or labels. They can also be copied and concatenated, one after another, into a single to-file member or label. If the to-file is a spooled file, then the copy command copies each member or label to a separate spooled file. If TOFILE(*PRINT) is specified, then all the members/labels are copied to a single spooled file, with the records for each member/label starting on a new page.

A single member or label, or multiple members or labels, can be copied to corresponding like-named to-file members or labels by specifying TOMBR(*FROMMBR), TOLABEL(*FROMMBR), or TOMBR(*FROMLABEL) depending on the copy command used. If the to-file is tape, you cannot specify this unless you are copying from a single from-file member or label. *FROMMBR is the default value for the TOMBR parameter on the CPYSRCF command, which copies the from-file members to like-named to-file members.

For more information:

For additional information, see the following topics:

- "How the copy function adds members to the to-file" on page 78
- "Allowed copy operations and parameters"

Allowed copy operations and parameters

This table shows the file types into which you can copy members or labels based on the source file type:

Diskette To:	Database To:
Database (physical file)	Database (physical file)
Diskette (Note 1)	Diskette
Tape (Note 2)	Tape (Note 2)
Printer	Printer
*PRINT	*PRINT

Notes:

1. The to-file must be spooled for diskette-to-diskette copy operations.
2. Multiple from-file members or labels can only be copied to a single tape file label.

This table shows the valid member or label parameters for copy commands:

Table 9. Valid Member or Label Parameters for Copy Commands

	FROMMBR[1]	FROMLABEL	TOMBR	TOLABEL
CPYF	X		X	
CPYFRMDKT		X	X	
CPYFRMQRYF			X	
CPYFRMTAP		X	X	
CPYSRCF	X		X	
CPYTODKT	X			X
CPYTOTAP	X			
CPYFRMIMPF	X		X	
CPYTOIMPF	X		X	

:

[1] FROMMBR is not a parameter on the CPYFRMQRYF command because the members to be queried are specified on the OPNQRYF command.

Copy all members or labels within a file

For database or diskette files, copy all members by specifying *ALL on the the FROMMBR or FROMLABEL parameter.

For diskette files, when you specify FROMLABEL(*ALL) on the CPYFRMDKT command and you specify a LABEL parameter value on an OVRDKTF command, only the single-file label identifier specified in the override is copied.

Copy only certain members or labels within a file

For database or diskette files, you first specify a generic name on the FROMMBR or FROMLABEL parameter. You then modify the generic name to indicate the starting character string that each member or label has in common, then follow it with an * (asterisk). For example, if you specified FROMMBR(ORD*), the copy command would copy all database members or diskette labels that start with ORD.

Note:

- If a generic name is specified for the FROMLABEL parameter on the CPYFRMDKT command and a LABEL parameter value is also specified on an Override Diskette File (OVRDKTF) command, the command copies only the single-file label identifier that you specified on the override.
- If you copy a generic set from a diskette, and a label *that is being copied* continues on another diskette volume, then the copy command copies all the affected labels on the continuation volume. This is also true when you copy all labels.

Specifying the label identifier or member name for the copy operation

If you specify TOMBR (*FIRST), the copy operation does not specify a label identifier. Therefore, you must specify a label identifier (LABEL parameter) either:

- In the device file on an OVRDKTF command (for a diskette file) OR
- On an OVRTAPF command (for a tape file)

If you specify the special value *FIRST, *DKTF, or *TAPF on the copy command, then the copy command uses the label from the device file description.

If the from-file is diskette or tape, the copy command uses the from-file label as the label for a diskette or tape to-file. If the to-file is a database file, the command uses the nonblank characters to the extreme right of the from-file label for the member name of the to-file. The command uses up to a maximum of either 10 characters or to the period at the extreme right in the from-file label. The copy operation uses only valid member names for a database to-file. It does not ensure that a to-file label is valid for tape or diskette, so a label identifier that is nonstandard or not valid may be used for the to-file.

If the from-file is a tape file that is not labeled, then a to-file member or label name is created that corresponds to the data file on the tape from-file in the form of CPYnnnnn, where nnnnn is the tape sequence number of the data file.

If you specify a tape or diskette label in the FROMMBR or TOMBR parameter, it can have a maximum length of 10 characters. If the label contains special characters or more than 10 characters, you must specify the label on one of the following commands:

- Create Tape File (CRTTAPF)
- Change Tape File (CHGTAPF)
- Override with Tape File (OVRTAPF)
- Create Diskette File (CRTDKTF)
- Change Diskette File (CHGDKTF)
- Override with Diskette File (OVRDKTF)

Special considerations for the Override Database File (OVRDBF), Override Diskette File (OVRDKTF), and Override Tape File (OVRTAPF) commands

For a database from-file or to-file, if a MBR parameter is specified on an OVRDBF (Override Database File) command, then the override member name is used instead of the value specified on the copy command. If the TOFILE parameter is specified with no MBR parameter value on the OVRDBF command, then the first member (in creation order) in the database file is used instead of the member

specified on the copy command. For a diskette or tape from-file or to-file, if a LABEL parameter is specified on an OVRDKTF or OVRTAPF command, respectively, the override label name is used instead of the label specified on the copy command.

If you copy multiple members or labels to corresponding like-named to-file members or labels, then you cannot use an override to a single to-file member or label unless you also override the from-file to a single member or label.

How the copy function adds members to the to-file

The copy function adds a member to the to-file when the member does not exist. The member name used is either the TOMBR parameter value from the copy command, or the member name that is specified in an override for the to-file.

If TOMBR(*FROMMBR) or TOMBR(*FROMLABEL) is specified on the copy command (and is not overridden), the from-file member names or label identifiers are used for the members added to the file.

If TOMBR(*FIRST) is specified on the copy command, or if there is an override that specifies a TOFILE parameter with no MBR parameter, then no member name is known. The copy function does not add a member in this case unless the following are true:

- You specified CRTFILE(*YES) on the copy command
- The copy function must create the to-file

Except for the CPYFRMQRYF command, when the copy function creates the to-file without a specific member name specified, the from-file name is used for the member that is added to the to-file. When using the CPYFRMQRYF command, the member added to the physical file that is created by the copy operation has the name specified by the TOMBR parameter. If you specify TOMBR(*FIRST), the to-file member has the same name as the to-file file name that is specified on the TOFILE parameter of the CPYFRMQRYF command. The copy command ignores the MBROPT parameter value when it creates the to-file, and adds records to the new file members.

If the from-file is a database file, the copy command uses the member text and SEU source type of the from-file member for the member that is added to the to-file. If the from-file is a device or inline data file, the copy command takes the text from message CPX0411; the SEU source type is TXT. If both the from-file and to-file are database source files, the SEU source type information in the added member will be the same as the from-file member. When it adds the to-file member, the copy command always assigns the SHARE(*NO) and EXPDATE(*NONE) attributes to the to-file member. The copy command also sets the creation date of the new member to the current system date (not the date when the from-file member was added).

When the copy command adds a member to a to-file that is a parent file, the constraint becomes established at that time.

Select the records to copy

The following topics show how you can use parameters on the copy commands to select only the specific records that you want to copy:

- "Select records using a specified record format name (RCDFMT Parameter)"
- "Select records by relative record numbers (FROMRCD and TORCD Parameters)"
- "Select records by record keys (FROMKEY and TOKEY Parameters)" on page 80
- "Select a specified number of records (NBRRCDS Parameter)" on page 85
- "Select records based on character content (INCCHAR Parameter)" on page 86
- "Select records based on field value (INCREL Parameter)" on page 87
- "Copy deleted records (COMPRESS Parameter)" on page 90

The copy command parameters for record selection (FROMRCD, TORCD, FROMKEY, TOKEY, INCCHAR, and INCREL) are not on the CPYFRMQRYF command because you select records on the OPNQRYF command.

See the *DB2 UDB for AS/400 Database Programming* book for details on record selection by using open query file. For a detailed description of all considerations for each parameter, see the *CL Programming* book.

Select records using a specified record format name (RCDFMT Parameter)

Note: You can use this parameter on the CPYF command only.

When you copy from a logical file to a physical file and the logical file has more than one record format, you must specify a record format name unless you specify FMTOPT(*NOCHK). If you use FMTOPT(*NOCHK), then you can specify RCDFMT(*ALL) to copy all from-file record formats to the to-file. The command uses this record format name to select records to copy.

This example shows how you can use the copy command to copy records from the logical file ORDFILL to the physical file INVOICE by using the record format ORDHDR:

```
CPYF   FROMFILE(DSTPRODLB/ORDFILL) +
  TOFILE(DSTPRODLB/INVOICE) RCDFMT(ORDHDR) +
  MBROPT(*ADD)
```

When you copy from a logical file that has more than one record format to a device file, you can specify either a single record format to be used or specify RCDFMT(*ALL) to copy using all the record formats. If the record formats have different lengths, the command pads the shorter records with blanks.

Select records by relative record numbers (FROMRCD and TORCD Parameters)

Note: You can use this parameter on the CPYF command only.

Relative record numbers can be specified for a copy from any file type except a keyed logical file. A keyed physical file can be copied in arrival order if relative record numbers are specified for the FROMRCD or TORCD parameter. Records can be copied:

- From a specified record number (FROMRCD parameter) to a specified record number (TORCD parameter) OR

- Until a specified number of records (NBRRCDS parameter) has been copied (see “Select a specified number of records (NBRRCDS Parameter)” on page 85)

If the command reaches the end of the file before it reaches the specified ending record number or number of records, the copy completes normally.

When a relative record number is specified, records are copied, starting with the specified relative record number, in the order in which they physically exist in the database file being copied from. This is true even if the physical file has a keyed sequence access path. You can use the COMPRESS parameter with the FROMRCD and TORCD parameters to further define which records you want to select for copying (see “Copy deleted records (COMPRESS Parameter)” on page 90).

If the from-file is a physical file or a logical file with an arrival sequence access path, the TORCD value is a relative record number that counts both the deleted and undeleted records ahead of it. If the from-file is a device file or inline data file, the TORCD value is a record number that includes only undeleted records (even for an I-format diskette file).

Deleted records retain their position among records that are not deleted. However these records do not necessarily retain their relative record number when they are copied if they are in the specified subset and COMPRESS(*NO) is specified. If you specify COMPRESS(*YES), the command skips the deleted records and does not copy them. In this case, when the record number that is specified (FROMRCD parameter) is a deleted record, copying starts with the first undeleted record that follows.

This example shows how you can use the command to copy records from relative record number 500 to relative record number 1000 in the file EMP1 to the file EMP1T.

```
CPYF    FROMFILE(PERSONNEL/EMP1) +
  TOFILE(TESTLIB1/EMP1T) MBROPT(*REPLACE) +
  FROMRCD(500) TORCD(1000)
```

Note: If you use record numbers to select records, you cannot use record keys (FROMKEY/TOKEY parameters) to select records on the same CPYF command.

For information about using the FROMRCD and TORCD parameters with distributed files, see the *DB2 Multisystem for AS/400* book.

Select records by record keys (FROMKEY and TOKEY Parameters)

Note: You can use this parameter on the CPYF command only.

You can specify record keys to copy only from a keyed database file. You can copy records:

- From a specified key value (FROMKEY parameter) to a specified key value (TOKEY parameter) OR
- Until a specified number of records (NBRRCDS parameter) is reached (see “Select a specified number of records (NBRRCDS Parameter)” on page 85

If the command reaches the end of the file before it reaches the specified ending key value or number of records, the copy completes normally.

If no record in the from-file member has a key that is a match with the FROMKEY value, but there is at least one record with a key greater than the specified value, the first record copied is the first record with a key greater than the FROMKEY value. If the specified key value is greater than any record in the member, the command sends an error message and does not copy the member.

You can specify *BLDKEY on the FROMKEY and TOKEY parameters to use a list of character and numeric values in their natural display form for the fields in a key. The command converts each element to the corresponding key field data type. The command then provides the **composite key** value (a key that is comprised of more than one field) to the database.

If you specify fewer values than the complete database key contains, the command builds a partial key and passes it to the database. If you specify more values than the database key contains, an ending error occurs. The command always applies values to the consecutive fields that are in the extreme left of the key so that it is impossible to skip key fields.

The command pads character fields on the right with blanks. The command adjusts numeric fields to the implied decimal point in the key field with the correct zero padding.

All regular rules for specifying numeric fields in an external character format apply. The command does not allow a floating-point value of *NAN (Not a Number).

See “Example: build-key function” on page 82 and “Example: using FROMKEY and TOKEY” on page 83 for specific coding examples.

It is also important to understand “Key string comparisons made by the copy operation” on page 82 in order to interpret various warning messages.

Note: If you use record keys to select records, you cannot use relative record numbers (FROMRCD/TORCD parameters) to select records on the same CPYF command.

You should not specify COMPRESS(*NO) when selecting records by record key from a keyed physical file. Because the keyed access path of a file does not contain deleted records, the copy command never copies them, so the compression is automatic.

Because deleted records are canceled in a copy by this method, it is also possible that the relative record numbers have changed in the new file, even if you have specified MBROPT(*REPLACE).

See the following topics for more information about specifying data for:

- “Variable-length fields” on page 83
- “Date, time, and timestamp fields” on page 83
- “Null-capable fields” on page 84
- “Different CCSIDs” on page 84
- “DBCS-graphic fields” on page 85

Key string comparisons made by the copy operation

The check made by the copy operation (when the TOKEY value is specified) is a logical character comparison between the key string for each record retrieved and the key string that is:

- Specified explicitly (using the first TOKEY parameter format) OR
- Built implicitly by the copy operation (that uses the list of values that are given)

A warning message is sent (but the copy operation continues) if this comparison gives different results than the ordering in which the database identifies the records in the keyed access path. The order may be different if:

- The key contains mixed ascending and descending fields
- The key contains fields for which a sort sequence other than *HEX is in effect OR
- The key contains any of the following DDS keywords:

  ABSVAL
  Absolute value

  ALTSEQ
  Alternative collating sequence

  ALWNULL
  Allow null

  DATFMT
  Date format (*MDY, *DMY, *YMD, *JUL, SAA *EUR, or SAA *USA)

  DIGIT Digit force

  SIGNED
  Signed numeric

  TIMFMT
  Time format (*USA)

  ZONE Zone force

If there are both ascending and descending fields in the file key, the first (the far left) key field determines whether the copy operation uses an ascending or descending key test to look for the last record to copy.

Using *BLDKEY is the easiest way to specify (and ensure correct padding) values for packed, binary, and floating-point fields.

Example: build-key function

An example of the build-key function is:

Key Field Number	Type	Length	Decimal Precision	Value
1	CHAR	6		KEN
2	ZONED	6	2	54.25
3	BINARY	4	1	10.1

You could specify the FROMKEY (or TOKEY) parameter as follows:

```
FROMKEY( 2  x'D2C5D5404040F0F0F5F4F2F50065')
```

Or, you could use the *BLDKEY value and specify the FROMKEY as follows:

```
FROMKEY(*BLDKEY    (KEN 54.25 10.1))
```

Another example using key fields 1 and 2 is:

```
FROMKEY(2  'KEN    005425')
```

Or, you can specify the *BLDKEY value:

```
FROMKEY(*BLDKEY    (KEN 54.25))
```

Example: using FROMKEY and TOKEY

In this example, the copy command copies records in the file EMP1 to the file EMP1T. EMP1T is a file in a test library. Because you only need a subset of the records, you specify a from-key value and a to-key value. Both are full key values. Note that a 1 specified in the FROMKEY and TOKEY parameters indicates the number of key fields to be used in searching the record keys, starting with the first key field.

```
CPYF   FROMFILE(PERSONNEL/EMP1) +
  TOFILE(TESTLIB1/EMP1T) MBROPT(*REPLACE) +
  FROMKEY(1 438872) TOKEY(1 810199)
```

All positions in a key value should be specified. If the value is shorter than the key field length, it will be padded on the right with zeros. Thus, a 5-position key field specified as FROMKEY(1 8) causes a search for a key equal to hex F800000000. If the key value contains blanks or special characters, you must enclose them in apostrophes.

Variable-length fields

When the number of key fields and a value are used to specify the FROMKEY or TOKEY parameter, the string should include the 2-byte length field for each variable-length key field. You must pad the variable-length key field with blanks so that keys following the variable-length key field are in the correct position. You can specify the data in hexadecimal format.

When you specify *BLDKEY on the FROMKEY or TOKEY parameter for variable-length key fields, specify the character string without the 2-byte length field. Only the amount of data that is entered for the key value is used for key comparisons. You can specify a zero-length string for variable-length key fields.

Date, time, and timestamp fields

When the number of key fields and a value are used to specify the FROMKEY or TOKEY parameter, no conversion of data occurs if the corresponding key field in the from-file is a date, time, or timestamp field. The user input string that you specify (including the separators) must be in the same format as the date, time, or timestamp field. If it is not, a file open error may occur, or the records copied may not be the desired result.

If *BLDKEY is specified for the FROMKEY or TOKEY parameter and the corresponding key field in the from-file is a date, time, or timestamp field, the system attempts to convert the user-input key field value to the format (and separators) of the from-file field. The following rules apply to the conversion:

- **If the from-field is a date key field,** the system first determines if the user-input key value is in the same format and has the same separator as specified in the current job under which the copy command is running. This can be *MDY, *DMY,

*YMD, or *JUL for the format and slash (/), hyphen (-), period (.), comma (,), or blank ( ) for the separator. If the user-input key value is not in the current job specified format and separator form, it determines if it is in one of the Systems Application Architecture (SAA) formats (*ISO, *USA, *EUR, or *JIS). It also determines if it is in a YYYYDDD form (no separator). If the system can determine the user-input key value is in one of these forms, the input string is converted to the actual format (and separator) of the from-file date field, which is used for the key comparison. If the user-input string format cannot be determined or the length or data value is not valid, the system issues a diagnostic message. You must left-justify the date portion of the user-input key value; it can contain trailing blanks.

- **If the from-field is a time key field,** the system first determines if the user-input key value is in the same format and has the same separator as specified in the current job under which the copy command is running. This may be HHMMSS for the format and colon (:), comma (,), period (.), or blank ( ) for the separator. If the user-input key value is not in the current job specified format and separator form, the system determines if it is in one of the SAA formats (*ISO, *USA, *EUR, or *JIS). If the system can determine the user-input key value is in one of these forms, the input string is converted to the actual format (and separator) of the from-file time field, which is used for the key comparison. If the user-input string format cannot be determined, or the length or data value is not valid, the system issues a diagnostic message. You must left-justify the time portion of the user-input key value; it can contain trailing blanks.
- **If the from-field is a timestamp key field,** the system first determines if the user-input key value is in the SAA format or YYYYMMDDHHMMSS form. If the system determines the user-input key value is in one of these forms, the input string is converted to the actual SAA timestamp format, which is used for the key comparison. If the user-input string format cannot be determined, or the length or data value is not valid, the system issues a diagnostic message. You must left-justify the timestamp portion of the user-input key value; it can contain trailing blanks.

Null-capable fields

When you use the number of key fields and a value to specify the FROMKEY or TOKEY parameter, the copy command ignores the null values. The command uses only the buffer default values for values that are actually null for the comparison.

When you specify *BLDKEY on the FROMKEY or TOKEY parameter, none of the *BLDKEY values can refer to a null-capable field. If they do, the system sends an error message.

Different CCSIDs

When you use the number of key fields and a value to specify the FROMKEY or TOKEY parameter, the copy command does not make any CCSID conversions to the input string.

When *BLDKEY is specified on the FROMKEY or TOKEY for character, DBCS-open, DBCS-either, or DBCS-only fields, the value specified is assumed to be in the CCSID of the process in which the copy command is running. The copy command converts each of these key values from the job CCSID to the CCSID of the from-file key field. If no conversion table is defined or an error occurs while converting the input key value, a message is sent and the copy operation ends. If

the value can be correctly converted, the converted value is used to build the key value that determines the first and last record to be copied.

DBCS-graphic fields

When the number of key fields and a value are used to specify the FROMKEY or TOKEY parameter, no conversions are done on the input string. The input string is used as is.

When you specify *BLDKEY on the FROMKEY or TOKEY for DBCS-graphic fields, you should enclose the DBCS data in shift-out and shift-in characters. The copy command assumes that the DBCS data is in the associated DBCS CCSID of the job CCSID. The shift-out and shift-in characters are removed before building the key. A message is sent and the copy operation ends:

- If the input string is not enclosed in shift-out and shift-in (SO-SI) characters OR
- The data cannot be converted to the DBCS CCSID of the from-file key field

Select a specified number of records (NBRRCDS Parameter)

Note: You can use this parameter on the following commands: CPYF, CPYFRMDKT, CPYFRMQRYF, CPYFRMTAP, CPYTODKT, and CPYTOTAP.

When you specify a FROMKEY or FROMRCD parameter, you can specify the number of records (NBRRCDS parameter) to be copied instead of the TOKEY or TORCD parameter. You cannot specify both the NBRRCDS and the TORCD or TOKEY parameters. The specified number of records is copied starting with the specified from-key value or from-record number.

You can specify the NBRRCDS parameter without specifying the FROMKEY or FROMRCD parameter. The copy command copies records by starting with the first record in the file. Note that the number of records specified is the number of records actually copied to the to-file, which includes

- Deleted records in the from-file if COMPRESS(*NO) is specified, but DOES NOT INCLUDE
- Records excluded by the INCCHAR and INCREL parameters

This example shows how you can use the copy command to copy 1000 records in the file EMP1 to the file EMP1T. The command copies records from the first member in EMP1 and replace the records in the first member in EMP1T.

```
CPYF   FROMFILE(PERSONNEL/EMP1) +
  TOFILE(TESTLIB1/EMP1T) MBROPT(*REPLACE) +
  NBRRCDS(1000)
```

You can also use the NBRRCDS parameter to examine a subset of records on a list:

```
CPYF   FROMFILE(PERSONNEL/EMP1) TOFILE(*PRINT) +
  FROMRCD(250) NBRRCDS(10) OUTFMT(*HEX)
```

When you successfully copy an open query file, the file position is unpredictable. If you want to run a different program with the same files or run another CPYFRMQRYF, you must position the file or close the file and open it with the same OPNQRYF command. You may position the file with the Position Database File (POSDBF) command. In some cases, you can use a high-level language program statement.

Select records based on character content (INCCHAR Parameter)

Note: You can use this parameter on the CPYF command only.

You can select records on the basis of the content of characters that start in a specific position in the record or field. You can use the INCCHAR parameter with the FROMKEY or FROMRCD parameter. You can select records first by their key value or relative record number, and then by characters in some position in the record or field.

You can test for any character string of 1 through 256 bytes. If the character string contains any special characters or blanks, you must enclose the entire string in apostrophes.

You can also specify *CT (contains) as the operator for the INCCHAR parameter. This specifies that the copy command should scan each record in the from-file for the selection character string. You can specify any valid starting position in the field or record for the start of the scan. The data will then be scanned from that position to the byte to the extreme right of the field or record.

If you specify both the INCCHAR and INCREL parameters, the copy command copies a record only if it satisfies both the INCCHAR and INCREL conditions.

This example shows how you can test for all records in the file DBIN that have an XXX starting in position 80. It then shows how you can copy these records to the file DKTOUT. Because this example includes testing for positions relative to the length of the whole record, you must specify *RCD on the INCCHAR parameter.

```
CPYF FROMFILE(DBIN) TOFILE(DKTOUT) +
  INCCHAR(*RCD 80 *EQ XXX)
```

If you were testing for an XXX in a position in a particular field in the record, you would specify the field name instead of *RCD, and the starting position of the character relative to the start of the field.

```
CPYF FROMFILE(DBIN) TOFILE(DKTOUT) +
  INCCHAR(FLDA 6 *EQ XXX)
```

A field name cannot be specified if RCDFMT(*ALL) is specified when copying from a multiple-format logical file, or if the from-file is a device file or inline data file.

See the following topics for additional information about specifying data for:

- “Variable-length fields”
- “Null-capable fields” on page 87
- “Different CCSIDs” on page 87
- “DBCS-graphic fields” on page 87

Variable-length fields

When you specify *RCD for the INCCHAR parameter, the starting position represents the position in the buffer. The 2-byte length field of variable-length fields must be considered when determining the position. Use single-byte blanks (X'40') to pad variable-length fields if the INCCHAR value spans multiple fields.

You can specify variable-length fields for the INCCHAR string when you specify a field name. The starting position represents the position in the data portion of the variable-length from-field value. The number of bytes that are compared is the number of bytes in the value that is specified for the INCCHAR string. If the actual data in the variable-length from-field is shorter than the value specified for the INCCHAR parameter, the from-field data is padded with single-byte blanks (X'40') for the comparison.

You cannot specify a zero-length string for the INCCHAR value.

Null-capable fields

The INCCHAR parameter allows null-capable character-field and null-capable DBCS-field names to be specified. However, any logical comparison with a null-field value tests as false, and the record is not copied. The copy command performs no special processing if the you specify the *RCD special value as the field name. The command only compares buffer default values for actual null values.

Different CCSIDs

When you specify *RCD for the INCCHAR parameter, the copy command does not perform any conversions on the input string. The command compares the byte string that you entered at the specified position in the record buffer of the from-file.

When you specify a field name, the command assumes that the input string is in the CCSID of the job in which the copy command runs. The input string is converted to the CCSID of the from-field. If no conversion table is defined or if an error occurs while converting the input string, a message is sent and the copy operation ends. If the command can correctly convert the value, the command uses the converted value for record selection.

DBCS-graphic fields

When you specify a graphic field for the INCCHAR parameter, you should enclose the DBCS data in shift-out and shift-in characters. The command assumes that the data is in the associated DBCS CCSID of the job CCSID. There must be a valid conversion to the field CCSID; otherwise, an error occurs. The shift-out and shift-in characters are removed before doing the comparison. The position specifies the DBCS character position in which to begin the comparison.

Select records based on field value (INCREL Parameter)

Note: You can use this parameter on the CPYF command only.

You use the INCREL parameter to select records for copying by testing for the value of an entire field. Unlike the INCCHAR parameter, you can use the INCREL parameter only when you are copying from a database file, and you can test for different values in different fields on one copy command.

You can use as many as 50 AND and OR relationships on one INCREL parameter. The OR relationship groups the AND relationships. For example, the following INCREL parameter essentially says this: If field FLDA is greater than 5 and field FLDB is less than 6, select the record. If FLDB is equal to 9 (FLDA is any value), select the record.

```
INCREL((*IF FLDA *GT 5) (*AND FLDB *LT 6) +
  (*OR FLDB *EQ 9))
```

The value you specify must be compatible with the field type. You must enclose each INCREL relational set in parentheses.

The value *IF must be specified as the first value in the first set of comparison values, if there is only one set or several sets of comparison values. If more than one set of comparison values are specified, either *AND or *OR must be specified as the first value in each set after the first set of values.

In the following discussion, an IF group refers to an IF set, optionally followed by one or more AND sets. An OR group refers to an OR set, optionally followed by one or more AND sets. All the comparisons specified in each group are done until a complete group, which is a single IF set or OR set having no AND sets following it, yields all true results. If at least one group has a true result, the copy command includes the record in the copied file.

The first set of comparison values (*IF field-name operator value) and any AND sets logically connected with the IF set are evaluated first. If the results in all of the sets in the IF group are true, the testing ends and the record is copied. If any of the results in the IF group are false and an OR group follows, another comparison begins. The command evaluates the OR set and any AND sets that follow it (up to the next OR set). If the results in the OR group are all true, the record is included. If any result is false and another OR group follows, the process continues until either an OR group is all true or until there are no more OR groups. If the results are not all true for any of the IF or OR groups, the record is excluded (not copied to the to-file).

If you specify both the INCCHAR and INCREL parameters, the copy command copies a record only if it satisfies both the INCCHAR and INCREL conditions.

You cannot specify the INCREL parameter if you specify RCDFMT(*ALL) when copying from a multiple-format logical file.

See the following for additional information on specifying data for:

- “Variable-length fields”
- “Date, time, and timestamp fields” on page 89
- “Null-capable fields” on page 89
- “Different CCSIDs” on page 90
- “DBCS-graphic fields” on page 90

Variable-length fields

You can use variable-length character fields for the INCREL parameter. Enter the character value without the 2-byte length field. The length of the data that is entered determines the number of bytes that are used for the comparison. If the actual data in the variable-length from-field is shorter than the value specified for the INCREL parameter, the from-field data is padded with single-byte blanks (X'40') for the comparison.

Date, time, and timestamp fields

The INCREL parameter allows date, time, and timestamp fields. The copy command compares the input field value chronologically to the value in the date, time, or timestamp field to determine if it should select the record. The system attempts to convert the input string and the actual field value to an internal form that is chronologically compared. These rules apply to the conversion:

- **If the from-field is a date field,** the system determines if the user-input field value is in the same format and has the same separator as specified in the current job under which the copy command is running. The format could be *MDY, *DMY, *YMD, or *JUL and could use a slash (/), hyphen (-), period (.), comma (,), or blank ( ) for the separator. If the user-input field value does not use the same format or separator form of the current job, the system determines if it is one of the SAA formats (*ISO, *USA, *EUR, OR *JIS) or if it is a YYYYDDD form with no separators. If the system determines that the user-input field value is one of these forms, it converts the input string to an internal form. The from-field is then converted to its internal form, and the comparison is made. If the user-input string format cannot be determined, or the length or data value is not valid, a diagnostic message is issued and the copy operation ends. You must left-justify the date portion of the user-input field value; it can contain trailing blanks.
- **If the from-field is a time field,** the system determines if the user-input field value is in the same format and has the same separator as specified in the current job under which the copy command is running. The format could be HHMMSS and have a colon (:), comma (,), period (.), or blank ( ) for the separator. If the user-input field value is not in the specified format and separator form of the current job, the system determines if it is in one of the SAA formats (*ISO, *USA, *EUR, or *JIS). If the system determines that the user-input key value is in one of these forms, it converts the input string to an internal form. The from-field is then converted to its internal form, and the chronological comparison is made. If the user-input string format cannot be determined or the length or data value is not valid, a diagnostic message is issued and the copy operation ends. You must left-justify the time portion of the user-input field value; it can contain trailing blanks.
- **If the from-field is a timestamp field,** the system first determines if the user-input field value is in the SAA format or YYYYMMDDHHMMSS form (no separators). If the system determines that the user-input field value is in one of these forms, it converts the input string to an internal form. The from-field is then converted to its internal form and the chronological comparison is made. If the user input string format cannot be determined, or the length or data value is not valid, a diagnostic message is issued and the copy operation ends. You must left-justify the timestamp portion of the user-input field value; it can contain trailing blanks.

Null-capable fields

The INCREL parameter allows a value of *NULL as input for a field value. You can use the *EQ and *NE operators with the *NULL value to test whether a field in a database file contains the null value or not. *EQ means that the value is null, and *NE means that the value is not null when you specify the *NULL value. The *NULL value is not limited to null-capable fields.

Different CCSIDs

The copy command assumes that the input string for character, DBCS-open, DBCS-either, or DBCS-only fields are in the CCSID of the job in which the copy command is running. The input string is converted to the CCSID of the from-field. If no conversion table is defined or an error occurs while converting the input string, a message is sent and the copy operation ends. If the copy command can correctly convert the value, it uses the converted value for record selection.

DBCS-graphic fields

When you specify a graphic field for the INCREL parameter, you should enclose the DBCS data in shift-out and shift-in characters. The copy command assumes that the data is in the associated DBCS CCSID of the job CCSID. There must be a valid conversion to the field CCSID. Otherwise, an error occurs. The shift-out and shift-in characters are removed before doing the comparison.

Copy deleted records (COMPRESS Parameter)

Note: You can use this parameter on the CPYF command only.

You can copy deleted and undeleted records from one physical file member to another by specifying COMPRESS(*NO) on a copy command.

You may want to copy deleted records to preserve the relative record numbers of records that are copied from the from-file. If you do not use COMPRESS(*NO), only records that are not deleted are copied from the from-file. There are "Requirements", "Restrictions", and more "Details" on page 91 about copying deleted records.

Requirements

To use COMPRESS(*NO), the following conditions must be true:

- The from-file and to-file must both be physical files.
- The from-file and to-file must both be the same type (either source or data).
- The from-file and to-file must either have identical record formats or you must specify FMTOPT(*NOCHK) to perform the copy.
- You must use all the following (default) parameter values on the copy command:
  - PRINT(*NONE)
  - INCCHAR(*NONE)
  - INCREL(*NONE)
  - SRCOPT(*SAME)
  - ERRLVL(0)

Restrictions

You cannot specify COMPRESS(*NO) for the following types of access paths over the to-file, including when the access path is contained in a logical file and is based on the to-file member:

- Unique keys (you specified the UNIQUE keyword in the DDS).

- Select/omit specifications without the DYNSLT keyword (in the DDS for the file), and immediate or delayed maintenance (MAINT(*IMMED) or MAINT(*DLY) specified on the CRTPF or CRTLF command).
- Floating-point key field or logical numeric key field (in the DDS for the file), and immediate or delayed maintenance (MAINT(*IMMED) or MAINT(*DLY) specified on the CRTPF or CRTLF command). Note that a logical numeric key field is one of the following:
  - A numeric key field in a logical file
  - A field specified as a *to* field on the JFLD keyword that has different attributes than in the based-on physical file
  - A field specified as a sequencing field on the JDUPSEQ keyword that has different attributes than in the based-on physical file

You cannot specify COMPRESS(*NO) for any of the following cases:

- If you use the JRNPF command to journal the to-file
- If the to-file member is in use or if any access path over the to-file member is in use
- If you specify an EOFDLY wait time for the from-file on an OVRDBF command.

Details

COMPRESS(*NO) may allow the system to copy more quickly because records are transferred in blocks, but this is *not always* true. Usually, the COMPRESS(*NO) function does not significantly affect performance. One of the factors you should consider before you specify COMPRESS(*NO) is that the internal system function that must be used to perform this type of copy invalidates any keyed access paths that use the to-file member before the records are copied and then rebuilds the access paths after the copy is complete. The run time and resource that are required to rebuild the keyed access paths may be larger than the performance benefit that is gained by copying deleted records.

If COMPRESS(*NO) is *not* specified, the system may still use the internal functions to perform the copy, but the choice of how the copy is performed is based on the number of records in the from-file and to-file members before the copy, and the number of keyed access paths over the to-file member.

If MBROPT(*REPLACE) is specified, all keyed access paths over the to-file member must be invalidated and rebuilt, so specifying COMPRESS(*NO) does not cause any additional overhead for rebuilding access paths.

If the from-file is a keyed physical file and neither a FROMRCD nor TORCD relative record number value is specified on the copy commands to force the file to be processed in arrival sequence, COMPRESS(*NO) has no meaning because a keyed access path never contains any deleted records.

Print records (PRINT, OUTFMT, and TOFILE(*PRINT) parameters)

Note: You can use the parameters described in this topic on the CPYF, CPYFRMDKT, CPYFRMQRYF, and CPYFRMTAP commands.

You can print a list of all records copied, all records excluded, or all records causing ERRLVL output errors. You do this by specifying PRINT special values on a copy command. You can specify one or more of these listings on a single copy command, using character or hexadecimal format.

You can also print an unformatted listing of records. See “Creating an unformatted print listing” on page 93 for more information.

Printing a list of all records copied:

To print a list of all of the records that you copied, specify TOFILE(*PRINT) on the copy command. The records are printed using the IBM-supplied printer file QSYSPRT.

Printing a list of excluded records:

Specify *EXCLD on the PRINT parameter to print a listing of only the records that you excluded from the copy. When you specify PRINT(*EXCLD), the records print in the from-file format.

Printing a list of copied records:

Specify *COPIED on the PRINT parameter to print a listing of only the records that you copied. When you specify PRINT(*COPIED) and MBROPT(*UPDADD), the records copied and the records updated appear on the same listing. A message follows each updated record that states that it was an update.

Printing a list of records that cause errors:

Specify *ERROR on the PRINT parameter to print a listing of the records that caused ERRLVL output errors. (The ERRLVL parameter still controls the number of recoverable errors that can occur.) See “Prevent errors when copying files” on page 104 for information on error recovery and the ERRLVL parameter. Only the number of records up to one (1) greater than the ERRLVL value that is specified are printed in the *ERROR listing. The listing is similar to the PRINT(*COPIED) and PRINT(*EXCLD) listings.

Selecting the format of the listing:

Use the OUTFMT parameter to specify whether your listing prints in character or hexadecimal format. The default value is *CHAR, and records print in character format. If you specify *HEX, records print in both character and hexadecimal format.

If you specify TOFILE(*PRINT), the OUTFMT parameter again specifies the format that is used to print the records.

When you specify PRINT(*EXCLD), the records print in the from-file format. All character data is in the CCSID specified in the from-file field. For TOFILE(*PRINT) and PRINT(*COPIED) listings, and when the to-file is a print file, character data is in the CCSID specified in the to-file fields.

Example:

In this example, all records that are not copied (or excluded records) are printed:

```
CPYF FROMFILE(DKTIN) TOFILE(LIB1/PF) +
     MBROPT(*ADD) INCCHAR(*RCD 80 *EQ X) +
     PRINT(*EXCLD)
```

The records print in character format.

Creating an unformatted print listing

If you want an unformatted print listing or if the from-file records should be formatted using first-character forms control (CTLCHAR(*FCFC), you must specify a program-described printer device file name. This file name can be QSYSPRT or user-defined (instead of *PRINT).

To format the from-file records using first-character forms control, specify CTLCHAR(*FCFC) on the Create Printer File (CRTPRTF), Change Printer File (CHGPRTF), or Override Printer File (OVRPRTF) command.

For copy commands where TOFILE(*PRINT) is specified with a PRINT parameter value of *COPIED, *EXCLD, or *ERROR (or any combination), the following limits apply:

- The QSYSPRT file must be spooled [SPOOL(*YES)]
- You must specify the QSYSPRT in the device file or on the OVRPRTF command, because separate print files open for each file requested.

All records are copied to a single spooled file, and the data for each member or label identifier copied begins on a new print page.

Copying between different database record formats (FMTOPT parameter)

(CPYF and CPYFRMQRYF commands)

When you copy from a database file to a database file, you must use the FMTOPT parameter if the record formats are not identical or if the files are different types (source or data). If either file is a device file or inline data file, the FMTOPT parameter does not apply. The records are truncated or padded with blanks or zeros when record lengths are different. A message is sent if the records are truncated.

For database files, when either FMTOPT(*CVTSRC) or FMTOPT(*NOCHK) is specified and the record data copied from any from-file record is not long enough to fill a to-file record, the extra bytes in the to-file record are set to a default value. If a default value other than *NULL is specified in the DDS (DFT keyword) for a field, that field is initialized to the specified default; otherwise, all numeric fields are initialized to zeros, all character fields are initialized to blanks, all date, time, and timestamp fields are initialized to the current system date and time. If *NULL is specified on the DFT keyword, only the default buffer value is used. A *NULL default is ignored.

If the from-file or to-file is a device file or an inline data file, copy automatically adds or deletes the source sequence number and date fields for each record copied.

If one file is a data file and the other a source file, you must specify FMTOPT(*CVTSRC) to perform the copy. The sequence number and date fields are added or deleted as appropriate and the data part of each record is copied without regard to the other field definitions in the file record formats. The SRCSEQ parameter can be used to control how the sequence numbers are created, provided SRCOPT(*SEQNBR) is also specified.

For database-to-database copies, you can reconcile any differences in record formats by specifying:

- *DROP to drop those fields in the from-file record format for which there are no fields of the same name in the to-file record format.
- *MAP to convert fields in the from-file to the attributes of like-named fields in the to-file and to fill extra fields in the to-file, that are not in the from-file, with their default values. The default values are:
  - The parameter value (including *NULL) for the DFT keyword, if specified for the field
  - Blanks (for character fields without the DFT keyword)
  - Zeros (for numeric fields without the DFT keyword)
  - Current date, time, or timestamp for those type fields without the DFT keyword

*MAP is required if fields with the same name are in different positions in the file record formats, even though these fields have the same attributes.

- *DROP and *MAP to drop fields in the from-file not named in the to-file and to convert remaining fields through mapping rules to fit the to-file fields that have different attributes or positions.
- *NOCHK to disregard the differences. Data is copied left to right directly from one file to the other. Null values are ignored. The copied records are either truncated or padded with default buffer values. Because no checking is done, fields in the to-file may contain data that is not valid for the field as defined.

Dropping and mapping fields are based on a comparison of field names. Unless all the fields in the from-file have the same name in the to-file, you must specify *DROP. If the names are the same, but the attributes or position in the record is different, you must specify *MAP. Dropped fields are not copied. There must be at least one like-named field in both record formats to do mapping.

When *MAP is specified, fields in the to-file record format that do not exist in the from-file record format are filled with their default values, as described earlier in this section. For fields that have the same name and attributes, the field in the from-file record format is mapped to the field with the same name in the to-file record format, even if their positions in the formats are different.

For example, the field CUSNO is the first field in the record format ORDHD, but it is the second field in record format ORDHD1. When the CUSNO field is copied with *MAP, it is mapped to the second field of ORDHD1.

Note: It is possible for files with large record formats (many fields) to have the same format level identifiers even though their formats may be slightly different. Problems can occur when copying these files if the record format names of the from-file and the to-file are the same. When copying such files using FMTOPT(*NONE) or FMTOPT(*MAP), it is recommended that the record format names of the from-file and the to-file be different.

Table 10 on page 95 summarizes the database-to-database copy operations for each value on the FMTOPT parameter.

Table 10. Database-to-Database Copy Operations

FMTOPT Parameter Values (see note 4)	**Database File Record Formats**				
	ALL Field Names in From-and To-Files Are the Same (like-named)	**SOME Field Names in From-and To-Files Are the Same**	**NO Field Names in Either File Are the Same**		
	Attributes and relative order also the same (see note 1)	Attributes and relative order not the same (see note 1)	Like-named fields have identical attributes and relative order (see note 1)	Not all like-named fields have identical attributes and relative order (see note 1)	
*NONE	Complete copy	Command ends	Command ends	Command ends	Command ends
*DROP	Complete copy (value ignored)	Command ends	If there are extra fields in the from-file, they are dropped, all others are copied. If there are extra fields in the to-file, the command ends. If there are extra fields in the from-file and in the to-file, the command ends.	Command ends	Command ends
*MAP (see note 2)	Complete copy (value ignored)	Complete copy (corresponding fields are mapped)	If there are extra fields in the from-file, the command ends. If there are extra fields in the to-file, they are filled, and the like-named fields are mapped. If there are extra fields in the to-file and the from-file, the command ends.		Command ends
*MAP and *DROP (see note 2)	Complete copy (value ignored)	Complete copy (corresponding fields are mapped)	Extra fields in the from-file are dropped; like-named fields are mapped; extra fields in the to-file are filled.		Command ends
*NOCHK	Complete copy (value ignored)	Complete copy (direct data transfer disregarding fields) (see note 3)			

Notes:

1. Field attributes include the data type (character, zoned, packed, binary or floating point), field length, decimal position (for numeric fields), date or time format (for date or time fields), null capability, CCSID, and whether the field has variable length or fixed length.
2. Mapping consists of converting the data in a from-file field to the attributes of the corresponding (like-named) to-file field. If the attributes of any corresponding fields are such that the data cannot be converted, the copy is ended.
3. The records are padded or truncated as necessary. Data in the from-file may not match the to-file record format.
4. Any other value specified for the FMTOPT parameter is ignored when the *CVTFLOAT value or the *NULLFLAGS value is specified (except the *CVTFLOAT and *NULLFLAGS values).

Specifying Data for Different Field Types and Attributes

Variable-Length Fields

FMTOPT(*MAP) can be used to map data between fixed- and variable-length fields and between variable-length fields with different maximum lengths.

When mapping a variable-length field with a length of zero to a:

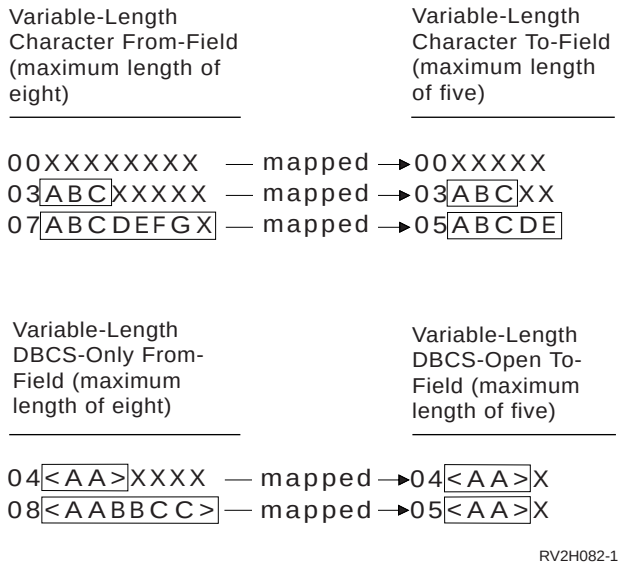
- variable-length to-field, the to-field length is set to zero.
- fixed-length to-field, the to-field is filled with single-byte blanks (X'40'), unless the to-field is a DBCS-only field. A DBCS-only to-field is set to X'4040's and surrounded by shift-out and shift-in (SO-SI) characters.

The following applies when the from-field does not have a length of zero and graphic fields are not being mapped to or from bracketed DBCS fields.

Mapping Variable-Length Fields to Variable-Length Fields

The length of a variable-length from-field is copied to a variable-length to-field when the from-field data length is less than or equal to the maximum length of the to-field. If the from-field data length is greater than the maximum length of the to-field, the data of the from-field is truncated to the maximum length of the to-field, and the to-field length is set to the maximum length. The data is truncated in a manner that ensures data integrity.

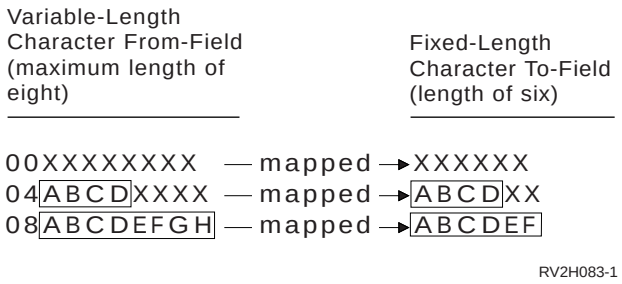
Note: In the examples, x represents a blank, < represents the shift-out character, and > represents the shift-in character. The 2-byte length is actually a binary number shown as a character to make the example readable.



Mapping Variable-Length Fields to Fixed-Length Fields

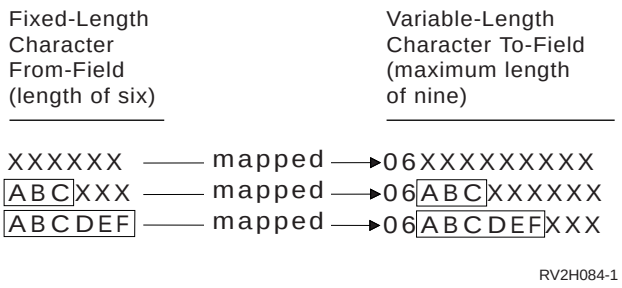
If the data length of the from-field is less than or equal to the to-field length, the data is copied to the fixed-length to-field and padded to ensure data integrity.

If the length of the from-field data is greater than the to-field length, the from-field data is copied to the to-field and truncated on the right in a manner that ensures data integrity.

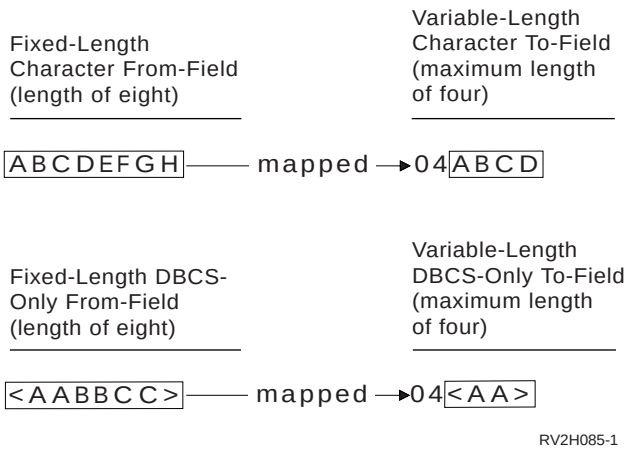


Mapping Fixed-Length Fields to Variable-Length Fields

If the to-field has a maximum length greater than or equal to the from-field length, the from-field data is copied to the data portion of the to-field and padded to the right with single-byte blanks. The to-field length is set to the length of the from-field length.



If the length of the from-field is greater than the maximum length of the variable-length to-field, the length portion of the variable-length to-field is set to the maximum length of the variable-length to-field. The data from the fixed-length from-field is copied to the data portion of the variable-length to-field and truncated on the right in a way that ensures data integrity.



Date, Time, and Timestamp Fields

FMTOPT(*MAP) or FMTOPT(*NOCHK) must be specified on the CPYF command if:

- The from-file is a database data file.
- The to-file is a physical data file.
- The record formats are not identical.

Corresponding date, time, and timestamp fields in the from-file and to-file must have the same format attribute and separator for the record formats to be identical. For the CPYFRMQRYF command, the same is true except that the open query file record format is used (rather than a from-file format).

When using FMTOPT(*NOCHK), record data is copied directly from left to right into the to-file without any regard to field types.

When using FMTOPT(*CVTSRC), data portions of records are directly copied from left to right into the to-file without any regard to the field types.

When using FMTOPT(*DROP), fields in the from-file but not in the to-file are dropped. If any like-named fields in the from-file and to-file are date, time, or timestamp fields, the corresponding field must be the same type, have the same format attribute and separator, and have the same relative position in the record format as the like-named field, otherwise FMTOPT(*MAP) may also be required.

FMTOPT(*MAP) allows copying between like date, time, and timestamp field types regardless of the format or separator. Also, copies from and to date, time, and timestamp fields are allowed from and to zoned-decimal or character field types, provided the lengths, formats, and values can be converted. FMTOPT(*MAP) is required in this case for conversion to the to-field type (format and separator, if it applies).

Table 11 on page 98 outlines the conversion possibilities for the date, time, and timestamp.

Table 11. Conversion Table

Data types	Forms	Allowable Field Length	Direction	Data Type	Formats	Allowable Field Length
Date	Any date format	6, 8, or 10	<-->	Date	Any	6, 8, or 10
Zoned	(MMDDYY)	6,0	<-->	Date	Any	6, 8, or 10
Zoned	(DDMMYY)	6,0	<-->	Date	Any	6, 8, or 10
Zoned	(YYMMDD)	6,0	<-->	Date	Any	6, 8, or 10
Zoned	(YYDDD)	5,0	<-->	Date	Any	6, 8, or 10
Character	(MMdDDdYY)	6 min	<-->	Date	Any	6, 8, or 10
Character	(DDdMMdYY)	6 min	<-->	Date	Any	6, 8, or 10
Character	(YYdMMdDD)	6 min	<-->	Date	Any	6, 8, or 10
Character	(YYdDDD)	6 min	<-->	Date	Any	6, 8, or 10
Character	(*USA)	6 min	---->	Date	Any	6, 8, or 10
Character	(*ISO)	6 min	---->	Date	Any	6, 8, or 10
Character	(*EUR)	6 min	---->	Date	Any	6, 8, or 10
Character	(*JIS)	6 min	---->	Date	Any	6, 8, or 10
Character	(YYYYDDD)	6 min	---->	Date	Any	6, 8, or 10
Time	Any time format	8	<-->	Time	Any	8
Zoned	(HHMMSS)	6,0	<-->	Time	Any	8
Character	(HHtMMtSS)	4 min	---->	Time	Any	8

Table 11. Conversion Table (continued)

Data types	Forms	Allowable Field Length	Direction	Data Type	Formats	Allowable Field Length
Character	(*USA)	4 min	---->	Time	Any	8
Character	(*ISO)	4 min	---->	Time	Any	8
Character	(*EUR)	4 min	---->	Time	Any	8
Character	(*JIS)	4 min	---->	Time	Any	8
Character	(HHtMMtSS)	8 min	<----	Time	Any	8
Timestamp	SAA format	26	<-->	Timestamp	SAA	26
Zoned	(YYYYMMDDHHMMSS)	14,0	<-->	Timestamp	SAA	26
Character	SAA format	14 min	---->	Timestamp	SAA	26
Character	(YYYYMMDDHHMMSS)	14 min	<-->	Timestamp	SAA	26

Note: In the format columns,

d = date separator value

t = time separator value

any = job formats or SAA formats

In the allowable field-length column, *min* means the specified length is the minimum required for a conversion attempt. Conversion errors may still occur if the length is not long enough for the desired or assumed format. Refer to the *DDS Reference* for more information on the date, time, and timestamp data types and keywords.

When converting a character field to a date, time, or timestamp field; FMTOPT(*MAP) is specified; and the corresponding from- and to-field names match; an attempt is made to determine what similar date form the character field is in. The following applies:

- ***For converting a character field to a date field,*** the minimum length required for the character field is 6. The system first determines if the character field data is in the same format and has the same separator as specified in the current job under which the copy command is running. This may be *MDY, *DMY, *YMD, or *JUL for the format and slash (/), hyphen (-), period (.), comma (,), or blank ( ) for the separator. If the character field is not in the current job specified format and separator form, it determines if it is in one of the SAA formats (*ISO, *USA, *EUR, or *JIS), or if it is in a YYYYDDD form (no separator). If the system determines the character field is in one of the these forms, it converts it to the date to-field. The date portion of the character field must be left justified and can contain trailing blanks.
- ***For converting a character field to a time field,*** the minimum length required for the character field is 4. The system first determines if the character field data is in the same format and has the same separator as specified in the current job under which the copy command is running. This may be *HMS for the format and colon (:), comma (,), period (.), or blank ( ) for the separator. If the character field is not in the current job specified format and separator form, the system determines if it is in one of the SAA formats (*ISO, *USA, *EUR, or *JIS). If the system determines the character field is in one of these forms, it converts it to the time to-field. The time portion of the character field must be left justified and can contain trailing blanks.
- ***For converting a character field to a timestamp field,*** the minimum length required for the character field is 14. The system first determines if the character field data is in one of the following:
  - SAA format
  - YYYYMMDDHHMMSS form

If the system determines the character field is in one of these forms, it converts it to the timestamp to-field. The timestamp portion of the character field must be left justified and can contain trailing blanks.

When converting a date, time, or timestamp field to a character field; FMTOPT(*MAP) is specified; and the corresponding from and to-file field names match; the system attempts to convert the date, time, or timestamp field into the form specified by the current job. The following applies:

- ***For converting a date field to a character field,*** the minimum length required for the character field is 6. The system first determines the date format and separator attribute of the current job under which the copy command is running. This may be *MDY, *DMY, *YMD, or *JUL for the format and slash (/), hyphen (-), period (.), comma (,), or blank ( ) for the separator. The date field is converted into the character field in the specified format of the current job. For character fields that are longer than required for the conversion, the data is left justified and trailing blanks are added.
- ***For converting a time field to a character field,*** the minimum length required for the character field is 8. The system first determines the time separator attribute of the current job under which the copy command is running. This may be colon (:), comma (,), period (.), or blank ( ). The time field is converted into the character field in the *HMS format (including the specified separator of the current job). For character fields that are longer than required for the conversion, the data is left justified and trailing blanks are added.
- ***For converting a timestamp field to a character field,*** the minimum length required for the character field is 14. The timestamp field is converted into the character field in the YYYYMMDDHHMMSS form (no separators). For character fields that are longer than required for the conversion, the data is left justified and trailing blanks are added.

When converting a zoned decimal field to a date, time, or timestamp field, FMTOPT(*MAP) is specified and the corresponding from- and to-field names match, the system assumes the zoned decimal field is in the form specified by the current job. The following applies:

- ***For converting a zoned decimal field to a date field,*** the system assumes the zoned decimal field data is in the same date format (no separators) as specified in the current job under which the copy command is running. This may be *MDY, *DMY, *YMD, or *JUL. The length of the zoned decimal field must be 5,0 (if the current job format is *JUL) or 6,0 (if the current job format is *MDY, *DMY, or *YMD). The system attempts to convert or copy it to the date to-field.
- ***For converting a zoned decimal field to a time field,*** the system assumes the zoned decimal field data is in the *HMS format (no separators). The length of the zoned decimal field must be 6,0. The system attempts to convert or copy it to the time to-field.
- ***For converting a zoned decimal field to a timestamp field,*** the system assumes the zoned decimal field data is in the YYYYMMDDHHMMSS form (no separators). The length of the zoned decimal field must be 14,0. The system attempts to convert or copy it to the timestamp to-field.

When converting a date, time, or timestamp field to a zoned decimal field, FMTOPT(*MAP) is specified and the corresponding from- and to-field names match, the system uses the current job specified form to determine what format the zoned decimal data should be in. The following applies:

- ***For converting a date field to a zoned decimal field,*** the system assumes the zoned decimal field data is to be in the same date format (no separators) as

specified in the current job under which the copy command is running. This may be *MDY, *DMY, *YMD, or *JUL. The length of the zoned decimal field must be 5,0 (if the current job format is *JUL) or 6,0 (if the current job format is *MDY, *DMY, or *YMD). The system attempts to convert or copy the date field to it.

- ***For converting a time field to a zoned decimal field,*** the system assumes the zoned decimal field data is to be in the *HMS format (no separators). The length of the zoned decimal field must be 6,0. The system attempts to convert or copy the time field to it.
- ***For converting a timestamp field to a zoned decimal field,*** the system assumes the zoned decimal field data is to be in the YYYYMMDDHHMMSS form (no separators). The length of the zoned decimal field must be 14,0. The system attempts to convert or copy the timestamp field to it.

Any conversion not successful because of a data value, data format, or data-length error causes an information message to be sent. The to-file field is set with its default value.

Null-Capable Fields

FMTOPT(*MAP) or FMTOPT(*NOCHK) must be specified on the CPYF command if:

- The from-file is a database data file.
- The to-file is a physical data file.
- The record formats are not identical.

For the record formats to be identical, corresponding fields in the from-file and to-file must both be null-capable or not null-capable. For the CPYFRMQRYF command, the same is true except that the open query file record format is used (rather than a from-file format).

When you use FMTOPT(*MAP):

- Null values are copied from null-capable from-file fields to null-capable to-file fields that are named alike. This copying can only happen if the field attributes and lengths are compatible.
- Fields that are not null-capable can also be copied from and to null-capable fields, provided the field attributes and lengths are compatible. The results to expect in the to-file field are:
  - Copying a null-capable field to a null-capable field

    Null values in the from-file field are copied to the to-file field. Values that are not null in the from-file field are also copied to the to-file field. For values that are not null in the from-file field that cause conversion errors during the copy, the default value of the to-file field is placed into the to-file field.
  - Copying a field that is not null capable to a null-capable field

    Values that are not null in the from-file field are copied to the to-file field. For values in the from-file field that cause conversion errors during the copy operation, the default value of the to-file field is placed into the to-file field.
  - Copying a null-capable field to a field that is not null capable

    Values that are not null in the from-file field are copied to the to-file field. If a conversion error occurs when copying values that are not null or the from-file field value is null, the to-file field default value is placed into the to-file.

When you use FMTOPT(*NONE), the null values in the from-file are copied to the to-file when copying a database file to a physical data file with identical record formats.

When you use FMTOPT(*DROP), the null values are copied.

When you use FMTOPT(*NOCHK) or FMTOPT(*CVTSRC), the record data is copied directly from left to right into the to-file without any regard to field types. Null values are not copied if *NOCHK or *CVTSRC is specified, because the record formats need not be identical. Either a user-specified or default value is copied to the to-file rather than a null value.

CCSIDs

When FMTOPT(*NOCHK) is specified, no CCSID conversions are done. Record data is copied directly from left to right into the to-file without any regard to field types or CCSIDs.

When FMTOPT(*MAP) is specified and a valid conversion is defined between the CCSID of the from-field and the CCSID of the to-file field, the character data is converted to the CCSID of the to-file field. However, if the CCSID of the from-file field or the CCSID of the to-file field is 65535, no conversions are done.

When FMTOPT(*NONE) is specified, the from-file and to-file attributes must be the same, unless one of the CCSIDs in corresponding fields is 65535.

For the CPYFRMQRYF command, the FMTOPT rules are the same except that the changed query format is used instead of a from-file format.

DBCS-Graphic Fields

When mapping graphic fields to bracketed DBCS fields, shift-out and shift-in characters are added around the DBCS data. When mapping from bracketed-DBCS fields to graphic fields, the shift-out and shift-in characters are removed. For variable-length fields, the graphic field length is expressed in the number of DBCS characters and the bracketed DBCS length is expressed in number of bytes (including the shift-out and shift-in characters). This difference is accounted for when mapping variable-length graphic fields to or from variable bracketed DBCS fields.

When using the CPYF command with FMTOPT(*MAP) to copy a DBCS-open field to a graphic field, a conversion error occurs if the DBCS-open field contains any SBCS data (including blanks). When copying to a graphic field, it may be desirable to ignore trailing SBCS blanks that follow valid DBCS data (in a DBCS-open field). This allows the copy operation to be done without a conversion error. This type of copy may be done using a combination of the OPNQRYF and CPYFRMQRYF commands. The OPNQRYF command is used to remove trailing single-byte blanks and place the data into a variable-length DBCS-open field. The CPYFRMQRYF command with FMTOPT(*MAP) specified is used to copy the variable-length DBCS-open field to the graphic field.

For example, assume the DBCS-open fields in the file named FILEO are copied into graphic fields in the file named FILEG. An additional file (FILEV) must be created.

The DDS for the original from-file FILEO:

```
******* ************** Beginning of data *****************************
            A          R FMT01
            A            FLD1          10O         CCSID(65535)
            A            FLD2           7O         CCSID(65535)
            A            FLD3          20A
******* ***************** End of data ********************************
```

DDS for FILEV: This file's format will be specified on the OPNQRYF command FORMAT parameter. The only difference from FILEO is that the DBCS-open fields to be converted to graphic fields are defined to be variable length.

```
******* ************** Beginning of data *****************************
            A          R FMT01
            A            FLD1          10O         VARLEN CCSID(65535)
            A            FLD2           7O         VARLEN CCSID(65535)
            A            FLD3          20A
******* ***************** End of data ********************************
```

DDS for the new file FILEG: The graphic fields are defined as fixed length; however, they could be made variable length, if desired.

```
******* ************** Beginning of data ************************
            A          R FMT01
            A            FLD1           4G         CCSID(65535)
            A            FLD2           3G         CCSID(65535)
            A            FLD3          20A
******* ***************** End of data ***************************
```

The following commands are used to copy the data from the DBCS-open fields in FILEO to the graphic fields in FILEG:

```
CHGJOB CCSID(65535)
OPNQRYF FILE((MYLIB/FILEO))
       FORMAT(MYLIB/FILEV *ONLY)
       MAPFLD((FLD1 '%STRIP(1/FLD1 *TRAIL)')
              (FLD2 '%STRIP(1/FLD2 *TRAIL)'))
CPYFRMQRYF FROMOPNID(FILEO) TOFILE(MYLIB/FILEG)
          MBROPT(*REPLACE) FMTOPT(*MAP)
```

Add or change source file sequence number and date fields (SRCOPT and SRCSEQ Parameters)

You can perform additions or changes to sequence number fields and date fields sequence number and date fields when you are:

- "Copying device source files to database source files"
- "Copying database source files to device source files" on page 104
- "Copying Database Source Files to Database Source Files" on page 104

Copying device source files to database source files

When you copy from a device source file to a database source file, the system adds sequence number fields and date fields at the start of the records. The system assigns the first record a sequence number of 1.00, the next 2.00, and so on, increasing in increments of 1.00. If more than 9999 records are copied, the sequence number wraps back to 1.00 and continues to increment unless you specify the SRCOPT and SRCSEQ parameters on the copy command.

If several copies to the same file are made with MBROPT(*ADD), you will have duplicate sequence numbers in the file. You can correct this using the Reorganize Physical File Member (RGZPFM) command.

Date fields are initialized to zeros.

When copying to or from a device, it is more efficient to use a device data file than a device source file. The copy function automatically adds or removes sequence number fields and date fields source sequence number and date fields as necessary.

Copying database source files to device source files

When you are copying to a device source file, the system removes the date fields and the sequence number fields from the start of the records.

When copying to or from a device, it is more efficient to use a device data file than a device source file. The copy function automatically adds or removes source sequence number fields and date fields as necessary.

Copying Database Source Files to Database Source Files

You can copy between database source files by using the CPYSRCF or CPYF command. The CPYSRCF command may be easier to use because the parameter defaults are better suited for copying database source files.

If you specify SRCOPT(*SEQNBR) to update the sequence numbers, the system considers the SRCSEQ parameter. The SRCSEQ parameter specifies the starting value that is assigned to the first record that is copied and the increment value. The defaults are 1.00 and 1.00. You can specify a whole number of no more than 4 digits or a fraction of no more than 2 digits for the starting value and the increment value. (You must use a decimal point for fractions.)

For example, if you specify SRCSEQ(100.50), then the records copied will have sequence numbers 100.00, 100.50, 101.00, 101.50, and so on.

Suppose that you have a file that contains more than 9999 records. Use a fractional increment value so that each record has a unique sequence number. If you specify a starting value of .01 and an increment value of .01, the maximum number of records copied with unique sequence numbers is 999 999. When the maximum sequence number is exceeded (9999.99), all remaining records on that copy are initialized to 9999.99. The system does not wrap back to 1.00.

If the database source file that you are copying to has only an arrival sequence access path, the records are always physically placed at the end of the file. (Because the file does not have a keyed sequence access path, you cannot insert records into the middle of the file keyed access path.)

Prevent errors when copying files

You can prevent many copy errors when you plan for certain conditions and situations ahead of time. The topics listed below provide guidance on the more common errors.

- “Limiting recoverable errors during copy” on page 105

Limiting recoverable errors during copy

When you copy to or from a database file or from a tape file, you can limit the number of recoverable errors that you accept before the copy ends. Use the ERRLVL parameter to specify this limit. This parameter applies to the following types of errors:

CPF4826
: Media error

CPF5026
: Duplicate key in the access path of this member. (Note: The copy command does not count CPF5026 as an ERRLVL error when you specify MBROPT(*UPDADD) on CPYF.)

CPF5027
: Record in use by another job. (Note: The copy command only counts CPF5027 as an ERRLVL error when you specify MBROPT(*UPDADD) on CPYF.)

CPF5029
: Data or key conversion error

CPF502D
: Referential integrity constraint violation

CPF502E
: Referential integrity constraints could not be validated

CPF5030
: Partial damage on member

CPF5034
: Duplicate key in the access path of another member

CPF5036
: Invalid length tape block read

CPF504B
: DataLink error

CPF504C
: DataLink preparation error

CPF5097
: *NAN (Not a Number) value not allowed in floating-point key field

The ERRLVL parameter specifies the maximum number of recoverable errors allowed for each label pair or each member copied. The value specified for ERRLVL indicates the total errors that are allowed on both the from-file and the to-file that are combined for each label pair or each member copied. Each time an error occurs, the following process runs:

1. The process increases the count for that label pair or that member by 1.

2. A message identifying the last good record that is read or written is printed on all copy lists if TOFILE(*PRINT), PRINT(*COPIED), or PRINT(*EXCLD) was specified.
3. The error record is printed if you specified PRINT(*ERROR).
4. Copying continues.
5. If the copy command completely copies the from-file member without exceeding the limit, the process resets the counter to 0, and the copy of the next member starts.
6. If the limit is exceeded during the copy of a member, copying ends and a message is sent, even if more records or members remain to be copied.

For a database from-file, including the open query file, the recoverable errors are:

- Those that occur when data is converted (mapped) AND
- Those caused by a damaged area on the disk (in auxiliary storage)

For a tape from-file, the recoverable errors are:

- A block length that is not valid AND
- A media-read operation from the tape volume on the device resulting in an error

For a physical to-file, the recoverable errors are:

- Those that occur when data is converted AND
- Those that occur when more than one of the same key is found

Any record that causes an error is not copied to the to-file. For a write error, the record is printed on a PRINT(*COPIED) and PRINT(*EXCLD) printout. A message then follows this printout. This message indicates that the record was not actually copied. If you specified PRINT(*ERROR), the command prints the records that caused write errors on the *ERROR listing. A message then indicates that an error occurred. For a read error, no record is available to be printed on the copy printouts (TOFILE(*PRINT), PRINT(*COPIED), PRINT(*EXCLD), or PRINT(*ERROR)). However, a message prints on all specified printouts that indicates that a record could not be read.

When the command cannot read a portion of the file from disk, partial object damage to the contents of a database file occurs. If a file is damaged in such a way, you can bypass the records that are in error by copying the good records and manually adding the records that were not copied because of the damage.

Regardless of the value of the ERRLVL parameter, recoverable errors always appear in the job log with a reply of “C” for “Cancel.”

For files that have constraint relationships, the ERRLVL parameter only affects the to-file. If you set the ERRLVL parameter to 0, the copy command does not copy into the file any record that causes the to-file to violate the constraint relationship. The copy operation ends. If ERRLVL is greater than 0, the copy command does not copy into the file any record that causes the to-file to violate the constraint relationship. However, the copy operation continues until enough violations (recoverable errors) have occurred so that the ERRLVL value has been reached. If this value is exceeded, the copy operation ends.

You can use the ERRLVL parameter to bring files with constraint relationships in check pending status back into non-check pending status. Do this by setting up the dependent to-file with constraints that are the same as the dependent from-file.

Then, use a CPYF command with the ERRLVL(*NOMAX) to copy all valid records. The to-file should not contain any records. The copy command does not insert into the to-file any records that it encounters from the from-file that would cause the to-file constraints to go to check pending status. With ERRLVL set to *NOMAX, the copy command processes all records in the from-file.

Other copy commands (CPYSRCF, CPYFRMTAP, CPYTOTAP, CPYFRMDKT, and CPYTODKT) end immediately if the systems signals one of the recoverable errors because there is no ERRLVL parameter for them.

Preventing date, time, and timestamp errors

For FMTOPT(*MAP), FROMKEY with *BLDKEY, TOKEY with *BLDKEY, and INCREL parameters, 2-digit year-date fields or values will be assumed to have:

- A century of 19 if the year is in the range from 40 to 99 OR
- A century of 20 if the year is in the range from 00 to 39

For example, 12/31/91 is considered December 31, 1991, while 12/31/38 is considered December 31, 2038.

However, any from-files containing 2-digit year-date fields with actual internal date values outside the range January 1, 1940 to December 31, 2039 cause input mapping errors, and the copy operation fails.

When FMTOPT(*MAP) is used to convert or copy a from-file field date value in a 4-digit year form to a 2-digit year form, the from-file field value must be within the range of January 1, 1940 to December 31, 2039. Otherwise, a mapping error occurs, and the copy command sets the to-file field with its default value.

Likewise, when using a 4-digit year date as a record selection input string on FROMKEY with *BLDKEY or TOKEY with *BLDKEY, the value must be within the same range if the corresponding from-file field is a date field with a 2-digit year-date. Otherwise, an error occurs. INCREL record selection is the exception to this rule, as 4-digit year date values outside this range may be used for corresponding 2-digit year-date fields.

See “Mapping considerations” for details about how to handle different field types and formats.

Mapping considerations

When mapping a character field to a date, time, or timestamp field and a format form is being used in the character field, leading zeros may be omitted from month, day, and hour parts. Microseconds may be truncated or omitted entirely in the character field.

For mapping to time fields, the seconds part (and corresponding separator) may be omitted from the character field.

For *USA form values, the AM or PM with a preceding blank is required. These rules are also true for date, time, or timestamp values that are entered when using FROMKEY with *BLDKEY, TOKEY with *BLDKEY, or INCREL parameters on the CPYF command. All other instances of date, time, and timestamp data require leading zeros when necessary and no truncation.

For both forms of the TOKEY parameter (*BLDKEY or non-*BLDKEY) the from-field data must be in a particular format for a chronological comparison to be made. For the date field, you must use the *ISO or *JIS format to make a chronological comparison. For the time fields, you must use the *HMS, *ISO, *EUR, or *JIS formats to make the chronological comparison. For any other formats of date or time fields (for date (*MDY, *DMY, *YMD, *JUL, *EUR, or *USA) or for time (*USA)), chronological comparisons are not possible because the TOKEY parameter performs a straight character string comparison. When you cannot make chronological comparisons, the system sends an informational message, and the copy operation continues.

When copying data into a file with date, time, or timestamp fields, and the from-file is a device file or FMTOPT(*NOCHK) or FMTOPT(*CVTSRC) has been specified, output mapping errors may occur if the data copied to a date, time, or timestamp field is not valid data for the to-file field format and separator attributes. You cannot copy the record if this occurs. If you use the CPYF or CPYFRMQRYF command, you can specify an error level other than zero (ERRLVL parameter) to bypass the record and continue the copy operation. When copying into date, time, or timestamp fields in these instances, it is important that the from-file data is valid for the to-file.

Preventing position errors

A position error occurs when the copy file function cannot locate the first record to copy in the from-file member. This can happen when using the CPYF, CPYSRCF, CPYTODKT, or CPYTOTAP commands. If any of the following conditions are true, you may receive a position error for the from-file member:

- You specified the FROMKEY parameter, and all records in the member are less than the FROMKEY value or the member is empty.
- You specified the FROMRCD parameter beyond the end of the member or the member is empty.
- The value of the from-file member position (the POSITION parameter of the OVRDBF command) is beyond the end of the member, is not valid for the access path of the from-file, or the member is empty.

If a member position error occurs, the member may not be added to the to-file, and no information about the member is added to the print lists.

If a member position error occurs during a copy operation that involves multiple members, the copy operation will continue with the next member.

If a member position error occurs for all members, a print list is not produced, and the to-file may not be created.

Preventing allocation errors

When a database file is copied, each from-file member is allocated with a shared-for-read (*SHRRD) lock state. When a device file is copied, the copy command allocates it with a shared-for-read (*SHRRD) lock state. The copy command allocates the member only while it copies it. A shared-for-read lock state lets other users read and update the file while you are copying it.

Generally, the member being copied to is allocated with a shared-for-update *SHRUPD) lock state. However, if you specify MBROPT(*REPLACE), the command allocates the member you are copying to with an exclusive (*EXCL) lock state, and the records in the to-file are removed

When you are copying one physical file to another, you can place stronger locks on the members to allow internal system functions to perform the copy.

- The command can allocate the from-file member with an exclusive-allow-read (*EXCLDRD) lock state.
- The command can allocate the to-file member with an exclusive (*EXCL) lock state.

The command requires these stronger locks depending on the type of copy you perform. If you cannot get these locks, run the copy command and specify a value of 1 (or any valid value other than 0) on the ERRLVL parameter. These values do not require the stronger locks.

There are many “Reasons for allocation errors”. For instance, you should not use functions that touch the to-file during the copy.

Reasons for allocation errors

If another job allocates a member with too strong a lock state, the copy operation may end with an error message. This is also true if the library containing the file is renamed during the copy operation.

When a copy command runs, the to-file may be locked (similar to an *EXCL lock with no time-out) so that no access is possible. Any attempt to use a function that must touch the to-file locks up the work station until the copy command completes. For instance, you should not use the following functions on a to-file that you are copying:

```
WRKACTJOB
    Option 11 (Work with Locks)
    Option 5  (Work with Job Member Locks)
    Option 8  (Work with Object Locks)
DSPDBR
DSPFD
DSPFFD
WRKJOB
    Option 12 (Work with Locks, if active)
    Option  5 (Work with Job Member Locks)
    F10 (Display Open Files, if active)
WRKLIB
    The library containing the to-file
DSPLIB
    The library containing the to-file
WRKOBJLCK
WRKRCDLCK
```

If you want to display any information about a to-file, you must anticipate the requirement and force the copy command to use block record-at-a-time operations by specifying ERRLVL(1).

If you anticipate that problems may arise because of this, you can preallocate the files and members using the Allocate Object (ALCOBJ) command. (See the *CL Programming* book for information about preallocating objects.)

Preventing copy errors that result from constraint relationships

A **constraint relationship** is a mechanism to ensure data integrity between a dependent file and a parent file. A constraint relationship exists between a dependent file and a parent file when every non-null foreign key value in the foreign key access path of the dependent file matches a parent key value in the parent key access path of the parent file. A physical data file may be a parent or dependent file. However, a source physical file may not be a parent or a dependent file.

The copy commands listed below allow the following relationships:

- CPYF - from-file or to-file could be a parent or dependent file
- CPYFRMQRYF - to-file could be a parent or dependent file
- CPYFRMTAP - to-file could be a parent or dependent file
- CPYTOTAP - from-file could be a parent or dependent file
- CPYFRMDKT - to-file could be a parent or dependent file
- CPYTODKT - from-file could be a parent or dependent file

See the following topics for more information about constraint relationships and copying files:

- "Copying files not in check-pending status"
- "Copying files in check pending status" on page 111

Copying files not in check-pending status

If the parent or dependent file has an established constraint relationship that is not in check-pending status, the following rules apply:

- If the from-file has an established constraint relationship, then you can copy all of the records from it whether it is a parent or dependent file.
- If the to-file has an established or enabled constraint relationship, then the following rules apply to keep the constraint relationship from entering check-pending status:
  - A parent file cannot have its member cleared of records.
  - A parent file cannot have more than one parent key value in the parent key access path of the same value (key must remain unique). That is, if the to-file is a parent file in a constraint relationship, then the copy does not allow duplicate key records to be copied into it.
  - A dependent file's foreign key values that are not null must always have a corresponding parent key value. That is, if the to-file is a dependent file in a constraint relationship, the copy operation does not allow non-null foreign key records that do not have a corresponding parent key record to be copied into the dependent file.

  The copy operation ensures that the data in the parent or dependent to-file is not damaged. Records may be copied to the to-file provided they do not cause the constraint relationship to go into check-pending status. If a user attempts to copy a record that does not meet the constraint relationship rules, the copy operation will end unless the ERRLVL parameter has been specified (CPYF and CPYFRMQRYF commands only) with a value greater than zero.

To circumvent the above rules, you can disable the involved constraints before the copy operation, perform the copy, and then re-enable the constraints. However, the file is in check-pending status if constraint rules are still not met.

Copying files in check pending status

If the parent or dependent file has an established constraint relationship that is in check-pending status, the following rules apply:

- If the from-file has an established constraint relationship in check pending, data access is restricted. If the from-file is a parent file, the command can read and copy data to the to-file. If the from-file is a dependent file, the command cannot read data to the to-file, and therefore cannot copy the data to the to-file.
- If the to-file has an established constraint relationship in check pending status, data access is restricted. If the to-file is a parent file, you can add new records (you can specify MBROPT(*ADD)). If the to-file is a parent file, you cannot clear the file (you cannot specify MBROPT(*REPLACE)). If the to-file is a dependent file, you cannot perform the copy regardless of which MBROPT parameter keyword you use.

To circumvent the above rules, you can disable the involved constraints before the copy operation, perform the copy, and then re-enable the constraints. However the file will be in check pending status if constraint rules are still not met.

Preventing copy errors related to your authority to files

The following table summarizes the authority that is required for the from-file and the to-file.

Table 12. Authority Required to Perform Copy Operation

	From-File	To-File
DDM file	*OBJOPR *READ	*OBJOPR[1] *ADD
Device file[2]	*OBJOPR *READ	*OBJOPR *READ
Logical file	*OBJOPR[3] *READ	Not allowed
Physical file	*OBJOPR *READ	*OBJOPR[1] *ADD

:

1 This is the authority required for MBROPT(*ADD). If MBROPT(*REPLACE) is specified, *OBJMGT and *DLT authority are also required. If MBROPT(*UPDADD) is specified, *UPD authority is also required.

2 *OBJOPR and *READ authority is also required for any devices used for the file.

3 Also requires *READ authority to the based-on physical file members for the logical file members copied.

If the to-file does not exist and CRTFILE(*YES) is specified so that the copy command creates the to-file, then you must have operational authority to the CRTPF command.

Improve copy performance

You can improve the performance of your copy operations by following these guidelines:

- “Avoid keyed sequence access paths” on page 112
- “Specify fewer parameters” on page 112

In addition, the DB2 Multisystem feature provides support for distributed files (that is, files that are spread across multiple AS/400 systems). When you copy distributed files, you should be familiar with the various factors that affect the performance of the copy command. You should be aware of restrictions that apply when you copy to and from distributed files.

For information about copying distributed files, see the *DB2 Multisystem for AS/400* book.

Avoid keyed sequence access paths

A copy that requires maintenance of a keyed sequence access path is slower than a copy from or to an arrival sequence access path. You can improve copy performance if you reorganize the from-file so that its arrival sequence is the same as its keyed sequence access path. You can also improve copy performance if you select records by using the FROMRCD or TORCD parameter so that the keyed sequence access path is not used.

Create fewer logical access paths over the to-file. This improves copy performance because the copy process does not need to update as many access paths.

The smaller the length of the records within the file, the faster the copy.

Specify fewer parameters

In general, you can improve copy performance if you specify fewer optional copy parameters. The following parameters affect the performance of the copy operation:

- INCCHAR
- INCREL
- ERRLVL
- FMTOPT
- SRCOPT
- PRINT

Using the COMPRESS function does not significantly affect performance. You should request COMPRESS(*NO) if you want deleted records in the to-file, for example, when the relative record numbers need to be identical.

Year 2000 support: date, time, and timestamp considerations

The CPYF and CPYFRMQRYF commands support the PACKED (P), ZONED (S), and CHARACTER (A) datatypes that have a DATFMT keyword specified in a logical file.

The copy converts data from or to a format implied by the length of a ZONED or PACKED field and the current job's DATFMT specification. Copy already supports ZONED fields with length 5,0 or 6,0 (depending on the current job DATFMT) from/to DATE fields.

FMTOPT(*MAP) allows copying between DATE field types and PACKED, ZONED, and CHARACTER field types in a logical or physical file provided the lengths, formats, and values can be converted. FMTOPT(*MAP) is required in these cases for conversion to the to-field type (and format and separator if it applies). There are

rules as to what form and length these field types must be in (dependent on the current job's DATFMT) for successful conversions.

New conversion possibilities exist for when you are:

- "Copying FROM logical file ZONED, CHARACTER, or PACKED field (with a DATFMT) TO a DATE field in a physical to-file".
- "Copying FROM or TO a ZONED or PACKED field (that has no DATFMT) TO or FROM a DATE type field" on page 115.

You also should know the system's restrictions on the conversions for Year 2000 support. See "Restrictions for Year 2000 support" on page 116 for more information.

The conversions involving CHARACTER fields from/to DATE fields do not change from existing support except that the logical file CHARACTER fields having a DATFMT specified are copied to a DATE field in a physical to-file. The system correctly converts the data.

Copying FROM logical file ZONED, CHARACTER, or PACKED field (with a DATFMT) TO a DATE field in a physical to-file

For these mappings, the format of the from-field is specified and is explicitly converted to the to-file DATE fields. These copies are single directional only: FROM the logical file ZONED, PACKED, or CHARACTER field TO a physical file DATE field.

The system allows century digit (C) in some of the forms. When the (C) value is 0, the system assumes the year is in the 1900s. When the (C) value is 1, the system assumes the year to be in the 2000s.

FLD TYPE	DATFMT	SPECIFIED FIELD LENGTH	COPY	DATA TYPE	FORMAT
ZONED	(*MY)	4,0	—>	DATE	(any)
ZONED	(*YM)	4,0	—>	DATE	(any)
ZONED	(*MYY)	6,0	—>	DATE	(any)
ZONED	(*YYM)	6,0	—>	DATE	(any)
ZONED [1]	(*JUL)	5,0	—>	DATE	(any)
ZONED [1]	(*MDY)	6,0	—>	DATE	(any)
ZONED [1]	(*DMY)	6,0	—>	DATE	(any)
ZONED [1]	(*YMD)	8,0	—>	DATE	(any)
ZONED [1]	(*ISO)	8,0	—>	DATE	(any)
ZONED [1]	(*EUR)	8,0	—>	DATE	(any)
ZONED [1]	(*JIS)	8,0	—>	DATE	(any)
ZONED [1]	(*USA)	8,0	—>	DATE	(any)
ZONED	(*LONGJUL)	7,0	—>	DATE	(any)
ZONED	(*CMDY)	7,0	—>	DATE	(any)
ZONED	(*CDMY)	7,0	—>	DATE	(any)
ZONED	(*CYMD)	7,0	—>	DATE	(any)
ZONED	(*MDYY)	8,0	—>	DATE	(any)

FLD TYPE	DATFMT	SPECIFIED FIELD LENGTH	COPY	DATA TYPE	FORMAT
ZONED	(*DMYY)	8,0	—>	DATE	(any)
ZONED	(*YYMD)	8,0	—>	DATE	(any)
CHAR	(*MY)	4	—>	DATE	(any)
CHAR	(*YM)	4	—>	DATE	(any)
CHAR	(*MYY)	6	—>	DATE	(any)
CHAR	(*YYM)	6	—>	DATE	(any)
CHAR [1]	(*JUL)	5	—>	DATE	(any)
CHAR [1]	(*MDY)	6	—>	DATE	(any)
CHAR [1]	(*DMY)	6	—>	DATE	(any)
CHAR [1]	(*YMD)	6	—>	DATE	(any)
CHAR [1]	(*ISO)	8	—>	DATE	(any)
CHAR	(*EUR)	8	—>	DATE	(any)
CHAR	(*JIS)	8	—>	DATE	(any)
CHAR [1]	(*USA)	8	—>	DATE	(any)
CHAR	(*LONGJUL)	7	—>	DATE	(any)
CHAR	(*CMDY)	7	—>	DATE	(any)
CHAR	(*CDMY)	7	—>	DATE	(any)
CHAR	(*CYMD)	7	—>	DATE	(any)
CHAR	(*MDYY)	8	—>	DATE	(any)
CHAR	(*DMYY)	8	—>	DATE	(any)
CHAR	(*YYMD)	8	—>	DATE	(any)
PACKED	(*MY)	4,0 5,0	—>	DATE	(any)
PACKED	(*YM)	4,0 5,0	—>	DATE	(any)
PACKED	(*YYM)	6,0 7,0	—>	DATE	(any)
PACKED	(*MYY)	6,0 7,0	—>	DATE	(any)
PACKED [1]	(*JUL)	5,0	—>	DATE	(any)
PACKED [1]	(*MDY)	6,0 7,0	—>	DATE	(any)
PACKED [1]	(*DMY)	6,0 7,0	—>	DATE	(any)
PACKED [1]	(*YMD)	6,0 7,0	—>	DATE	(any)
PACKED [1]	(*ISO)	8,0 9,0	—>	DATE	(any)
PACKED [1]	(*EUR)	8,0 9,0	—>	DATE	(any)
PACKED [1]	(*JIS)	8,0 9,0	—>	DATE	(any)
PACKED [1]	(*USA)	8,0 9,0	—>	DATE	(any)
PACKED	(*LONGJUL)	7,0	—>	DATE	(any)
PACKED	(*CMDY)	7,0	—>	DATE	(any)
PACKED	(*CDMY)	7,0	—>	DATE	(any)
PACKED	(*CYMD)	7,0	—>	DATE	(any)
PACKED	(*MDYY)	8,0 9,0	—>	DATE	(any)
PACKED	(*DMYY)	8,0 9,0	—>	DATE	(any)

FLD TYPE	DATFMT	SPECIFIED FIELD LENGTH	COPY	DATA TYPE	FORMAT
PACKED	(*YYMD)	8,0 9,0	—>	DATE	(any)

Notes:

[1] The DATFMTs in the logical file for these fields may not have actually been specified. If the DATFMT is not specified in the logical file, it will become the DATFMT specified on the underlying physical file Date field's DATFMT. If the DATFMT specified in the logical file was *JOB, it will become the actual DATFMT of the job.

Also note: in the FORMAT column, **(any)** means that any of the job formats or Systems Application Architecture (SAA) formats may be specified.

Copying FROM or TO a ZONED or PACKED field (that has no DATFMT) TO or FROM a DATE type field

FLD TYPE	ASSUMED FORM FOR DATFMT & LENGTH [1]	CURRENT JOB DATFMT	NUMERIC FIELD LENGTH	COPY	DATA TYPE	FORMAT
ZONED	(MMYY)	*MDY, *DMY	4,0	<—>	DATE	(any)
ZONED	(YYMM)	*YMD	4,0	<—>	DATE	(any)
ZONED [2]	(YYDDD)	*JUL	5,0	<—>	DATE	(any)
ZONED [2]	(MMDDYY)	*MDY	6,0	<—>	DATE	(any)
ZONED [2]	(DDMMYY)	*DMY	6,0	<—>	DATE	(any)
ZONED [2]	(YYMMDD)	*YMD	6,0	<—>	DATE	(any)
ZONED	(CMMDDY)	*MDY	7,0	<—>	DATE	(any)
ZONED [1]	(CDDMMYY)	*DMY	7,0	<—>	DATE	(any)
ZONED	(CYYMMDD)	*YMD	7,0	<—>	DATE	(any)
ZONED	(YYYYDDD)	*JUL	7,0	<—>	DATE	(any)
ZONED	(MMDDYYYY)	*MDY	8,0	<—>	DATE	(any)
ZONED	(DDMMYYYY)	*DMY	8,0	<—>	DATE	(any)
ZONED	(YYYYMMDD)	*YMD	8,0	<—>	DATE	(any)
PACKED	(MMYY)	*MDY, *DMY	4,0 5,0	<—>	DATE	(any)
PACKED	(YYMM)	*YMD	4,0 5,0	<—>	DATE	(any)
PACKED	(YYDDD)	*JUL	5,0	<—>	DATE	(any)
PACKED	(MMDDYY)	*MDY	6,0	<—>	DATE	(any)
PACKED	(DDMMYY)	*DMY	6,0	<—>	DATE	(any)
PACKED	(YYMMDD)	*YMD	6,0	<—>	DATE	(any)
PACKED	(CMMDDYY)	*MDY	7,0	<—>	DATE	(any)
PACKED	(CDDMMYY	*DMY	7,0	<—>	DATE	(any)
PACKED	(CYYMMDD)	*YMD	7,0	<—>	DATE	(any)
PACKED	(YYYYDDD)	*JUL	7,0	<—>	DATE	(any)
PACKED	(MMDDYYYY)	*MDY	8,0 9,0	<—>	DATE	(any)
PACKED	(DDMMYYYY)	*DMY	8,0 9,0	<—>	DATE	(any)
PACKED	(YYYYMMDD)	*YMD	8,0 9,0	<—>	DATE	(any)

FLD TYPE	ASSUMED FORM FOR DATFMT & LENGTH [1]	CURRENT JOB DATFMT	NUMERIC FIELD LENGTH	COPY	DATA TYPE	FORMAT

Notes:

[1] When copying from a PACKED or ZONED to a DATE, the assumed form is the form the copy will expect the data to be in. When copying from DATE to a PACKED or ZONED field the assumed form is the form copy will attempt to convert the data to.

[2] These conversions are already supported.

When converting/copying a ZONED field (with no DATFMT) from/to a DATE field (FMTOPT(*MAP) that is specified, the corresponding from and to-field names matching the system assumes the ZONED field is to be in a form determined from the current job DATFMT value and the ZONED field length (see table for specifics).

Similarly, when converting/copying a PACKED field (with no DATFMT) from/to DATE field (FMTOPT(*MAP) specified and the corresponding from and to-field names match, the system assumes the PACKED field is to be in a form determined from the current job DATFMT value and the PACKED field length (again, see table for specifics).

For the new DATFMTs that have a 'century guard digit', the system allows the values 0-9. 0 is for the year range 1900 through 1999, 1 is for 2000 through 2099, 2 is for 210 through 2199, and so forth, up to 9 for 2800 through 2899. The formats allowing 'century guard digit' are *CDMY, *CMDY, and *CYMD.

For the new DATFMTs that have no 'day' portion, *MY, *YM, *MYY, and *YYM, the day is assumed to be the first day of the month. For conversions to one of these DATFMTs from a DATFMT that has a 'day' portion, the 'day' value is removed.

For conversions from the no 'day' DATFMT to a DATFMT having a day portion, the 'day' value becomes the first day of the month. For example, the *YYMD value '19971231' becomes '199712' when converted to *YYM. When converted back, '199712' becomes '19971201'.

Restrictions for Year 2000 support

Record selection (FRMKEY, TOKEY, INCCHAR, and INCREL parameters) for CPY is not enhanced for the PACKED, ZONED, and CHARACTER data types having the DATFMT keyword. They are treated as their actual field type indicates, and a DATFMT specified on them is ignored for these parameters.

Likewise, when copying a logical file with PACKED, ZONED, or CHARACTER fields having the DATFMT specified to a like type PACKED, ZONED, or CHARACTER physical file field, the DATFMT on the from-field is ignored. No DATE conversions take place in these instances.

For ZONED and PACKED fields, if the length is not valid for the current job's DATFMT and assumed form, copy file diagnostic messages CPF2960 and CPF2963 is issued followed by a CPF2817 escape message.

If the length of the field is valid for the current job's DATFMT, the system attempts to convert/copy it from or to the DATE field. The system sends a CPF2958 message and the to-field is set with its default value:

- If the field value is incorrect (such as 13 for the month portion of *MDY form) or
- If a mapping error occurred because the data is not in the assumed form for the PACKED or ZONED field

The default value may be NULL, some user-defined value, or the default data-type value.

Copying complex objects

You can copy from and to files that contain user-defined functions (UDFs), user-defined types (UDTs), DataLinks (DLs), and large objects (LOBs). This topic describes AS/400 data management support for these objects.

Copying files that contain user-defined functions

You can specify CRTFILE(*YES) on the CPYF and CPYFRMQRYF commands when you copy files that contain user-defined functions (UDFs). UDFs do not get created with the new to-file.

You cannot copy DDM files that contain user-defined functions to AS/400 systems running at Version 4 Release 3 or earlier.

Copying files that contain user-defined types

You can specify CRTFILE(*YES) on the CPYF and CPYFRMQRYF commands when you copy files that contain user-defined types (UDTs). If the from-file is an SQL table, view, or index that contains a UDT, these commands create an SQL table.

You can copy UDTs to other UDTs using FMTOPT(*MAP), provided that you are copying from and to the same (identical) UDT. You can also copy from a non-UDT to a UDT, provided that the source type is compatible. Data mapping is not allowed if you are copying between UDTs that are not identical. Also, data mapping is not allowed if you are copying from a UDT to a non-UDT.

You cannot copy DDM files that contain user-defined types to AS/400 systems running at Version 4 Release 3 or earlier.

Copying files that contain DataLinks

You can specify CRTFILE(*YES) on the CPYF and CPYFRMQRYF commands when you copy files that contain DataLinks (DLs). If the from-file is an SQL table, view, or index that contains a DL, these commands create an SQL table.

You cannot copy DDM files that contain DataLinks to AS/400 systems running at Version 4 Release 3 or earlier.

DLs can be mapped only to other DLs. Therefore, if you specify *NONE, *MAP, or *DROP on the FMTOPT parameter, the from-file and to-file must have corresponding DLs. Truncation is not allowed. Shorter DLs, however, can be converted to longer DLs.

A file can be linked only once on a system. Therefore, a copy that will perform mapping or that requires the formats to be identical (that is, *NONE, *MAP, or *DROP is specified on the FMTOPT parameter) will not be successful if corresponding from-file and to-file fields are both FILE LINK CONTROL. Copies that are performed using the *NOCHK parameter option are not restricted, but errors will occur if a DL that references a linked file is copied to a DL that is FILE LINK CONTROL.

When you specify CRTFILE(*YES) on the CPYF or CPYFRMQRYF command, and the from-file contains a FILE LINK CONTROL DL field, the following statements are true, depending on how you specify the FMTOPT parameter:

- If you specify *NONE, *MAP, or *DROP on the FMTOPT parameter, the file is created, but an error message is issued and no I/O is performed.
- If you specify *NOCHK or *CVTSRC on the FMTOPT parameter, the file is created and I/O is attempted. The I/O will be unsuccessful for any records that contain a valid LINK.

The following table shows LINK scenarios associated with the CPYF command when different FMTOPT values are used.

LINK status for from-field to to-field when FMTOPT parameter is *MAP or *NONE	**How linking is performed**
FILE LINK CONTROL to FILE LINK CONTROL	Not allowed. Files can be linked only once.
NO LINK CONTROL to FILE LINK CONTROL (with no truncation)	Linking is performed.
FILE LINK CONTROL to NO LINK CONTROL (with no truncation)	No linking is performed.
NO LINK CONTROL to NO LINK CONTROL (with no truncation)	No linking is performed.

Copying files that contain large objects

You can specify CRTFILE(*YES) on the CPYF and CPYFRMQRYF commands when you copy files that contain large objects (LOBs). If the from-file is an SQL table, view, or index that contains a LOB, these commands create an SQL table.

AS/400 supports three large object data types: Binary Large OBjects (BLOBs), single-byte or mixed Character Large OBjects (CLOBs), and Double-Byte Character Large OBjects (DBCLOBs). When you copy files that contain these objects using the Copy File (CPYF) command, you should consider the following restrictions and requirements:

- LOB data is not copied when you copy from and to device files, when you copy to *PRINT, or when you specify values of *NOCHK or *CVTSRC on the FMTOPT parameter. In these cases, only the default buffer value for the LOB field is copied, including ″*POINTER″. This is true even when you copy a file that contains a LOB field to an identical file. Valid LOB data is copied only when you have specified *NONE, *MAP, or *DROP on the FMTOPT parameter.
- LOB data is not copied when you copy to a tape or diskette. In these cases, only the buffer value (including ″*POINTER″) is written to the tape or diskette. In addition, if you copy from the tape or diskette back to the same file, you will receive errors; this is because the file contains only the ″*POINTER″ value and not a valid pointer to actual LOB data.

- When you specify *UPDADD on the MBROPT parameter of the CPYF command, the to-file can contain a LOB field. LOB fields are also updated when duplicate key errors are encountered.
- When you specify *CVTFLOAT or *CVTNULLS on the FMTOPT parameter of the CPYF command, the to-file cannot contain a LOB field.
- If you want to print a file that contains LOB fields, specify *PRINT on the TOFILE parameter of the CPYF command. ″*POINTER″ will appear in the print listing in place of the LOB field data, and other non-LOB field data will also appear in the listing. If you have not specified *PRINT on the TOFILE parameter and you specified *COPIED, *EXCLUDE, or *ERROR on the PRINT parameter, then you must specify *NOCHK or *CVTSRC on the FMTOPT parameter for the copy to be allowed.
- You cannot specify LOB fields on the INCCHAR and INCREL parameters. You can specify *RCD or *FLD on the INCCHAR parameter, but only the fixed buffer length is compared, and not any actual LOB data.
- You cannot copy DDM files that contain LOB fields to AS/400 systems running at Version 4 Release 3 or earlier.

The following tables show how LOBs are mapped to other data types during copy operations. The first table shows the mapping when both fields contain LOB field types. In the tables, consider the following guidelines:

- The mapping of LOBs from and to DATE or TIME types is not allowed.
- These mappings are valid only for FMTOPT(*MAP) except where noted.
- There are similar data restrictions for large objects as those for normal character data (single-byte, mixed, and double-byte).

Table 13. From-file and to-file mapping when both fields are large objects

Field A type	Field B type	Allowed and copy direction	Data CCSID or attributes		CCSIDs	Conversion translation performed
			Field A	Field B		
BLOB	BLOB	Y* <—>	65535	65535	Same	No
CLOB	CLOB	Y* <—>	Character	Character	Same	No
CLOB	CLOB	Y* <—>	Open	Open	Same	No
DBCLOB	DBCLOB	Y* <—>	Graphic	Graphic	Same	No
DBCLOB	DBCLOB	Y* <—>	UCS2	UCS2	Same	No
CLOB	CLOB	Y <—>	Character	Character	Different	Yes
CLOB	CLOB	Y <—>	Open	Open	Different	Yes
DBCLOB	DBCLOB	Y <—>	Graphic	Graphic	Different	Yes
DBCLOB	DBCLOB	Y <—>	UCS2	UCS2	Different	Yes
CLOB	CLOB	Y <—>	Character	Open	Different	Yes
CLOB	DBCLOB	N	Character	Graphic	Different	—
CLOB	DBCLOB	Y <—>	Open	Graphic	Different	Yes
CLOB	DBCLOB	Y <—>	Character	UCS2	Different	Yes
CLOB	DBCLOB	Y <—>	Open	UCS2	Different	Yes
DBCLOB	DBCLOB	Y <—>	Graphic	UCS2	Different	Yes
BLOB	CLOB	Y <—>	65535	Character	Different	No
BLOB	CLOB	Y <—>	65535	Open	Different	No
BLOB	DBCLOB	N	65535	Graphic	Different	—

Table 13. From-file and to-file mapping when both fields are large objects (continued)

Field A type	Field B type	Allowed and copy direction	Data CCSID or attributes		CCSIDs	Conversion translation performed
			Field A	Field B		
BLOB	DBCLOB	N	65535	UCS2	Different	—
Note: * These mappings are valid for FMTOPT(*MAP), FMTOPT(*NONE), and FMTOPT(*DROP).						

The second table shows the mapping between fixed-length data types and large objects.

Table 14. From-file and to-file mapping between fixed-length data types and large objects

Field A type	Field B type	Allowed and copy direction	Data CCSID or attributes		CCSIDs	Conversion translation performed
			Field A	Field B		
CHAR	BLOB	Y <—>	Character	65535	Different	No
Open	BLOB	Y <—>	Open	65535	Different	No
Either	BLOB	Y <—>	Either	65535	Different	No
Only	BLOB	Y <—>	Only	65535	Different	No
Graphic	BLOB	N	Graphic	65535	Different	—
UCS2	BLOB	N	UCS2	65535	Different	—
Character	CLOB	Y <—>	Character	Character	Same/Different	No/Yes
Open	CLOB	Y <—>	Open	Character	Different	Yes
Either	CLOB	Y <—>	Either	Character	Different	Yes
Only	CLOB	Y <—>	Only	Character	Different	Yes
Graphic	CLOB	N	Graphic	Character	Different	—
UCS2	CLOB	Y <—>	UCS2	Character	Different	Yes
Character	CLOB	Y <—>	Character	Open	Different	Yes
Open	CLOB	Y <—>	Open	Open	Same/Different	No/Yes
Either	CLOB	Y <—>	Either	Open	Different	Yes
Only	CLOB	Y <—>	Only	Open	Different	Yes
Graphic	CLOB	Y <—>	Graphic	Open	Different	Yes
UCS2	CLOB	Y <—>	UCS2	Open	Different	Yes
Character	DBCLOB	N	Character	Graphic	Different	–
Open	DBCLOB	Y <—>	Open	Graphic	Different	Yes
Either	DBCLOB	Y <—>	Either	Graphic	Different	Yes
Only	DBCLOB	Y <—>	Only	Graphic	Different	Yes
Graphic	DBCLOB	Y <—>	Graphic	Graphic	Same/Different	No/Yes
UCS2	DBCLOB	Y <—>	UCS2	Graphic	Different	Yes
Character	DBCLOB	Y <—>	¬65535	UCS2	Different	Yes
Open	DBCLOB	Y <—>	¬65535	UCS2	Different	Yes
Either	DBCLOB	Y <—>	¬65535	UCS2	Different	Yes
Only	DBCLOB	Y <—>	¬65535	UCS2	Different	Yes
Graphic	DBCLOB	Y <—>	Graphic	UCS2	Different	Yes
UCS2	DBCLOB	Y <—>	UCS2	UCS2	Same/Different	No/Yes
Character	DBCLOB	N	65535	UCS2	Different	–

Table 14. From-file and to-file mapping between fixed-length data types and large objects (continued)

Field A type	Field B type	Allowed and copy direction	Data CCSID or attributes		CCSIDs	Conversion translation performed
			Field A	Field B		
Open	DBCLOB	N	65535	UCS2	Different	–
Either	DBCLOB	N	65535	UCS2	Different	–
Only	DBCLOB	N	65535	UCS2	Different	–

The second table shows the mapping variable-length data types and large object.

Table 15. From-file and to-file mapping between variable-length data types and large objects

Field A type	Field B type	Allowed and copy direction	Data CCSID or attributes		CCSIDs	Conversion translation performed
			Field A	Field B		
VARCHAR	BLOB	Y <—>	Character	65535	Different	No
VAROPEN	BLOB	Y <—>	Open	65535	Different	No
VAREITH	BLOB	Y <—>	Either	65535	Different	No
VARONLY	BLOB	Y <—>	Only	65535	Different	No
VARGRPH	BLOB	N	Graphic	65535	Different	—
VARUCS2	BLOB	N	UCS2	65535	Different	—
VARCHAR	CLOB	Y <—>	Character	Character	Same/Different	No/Yes
VAROPEN	CLOB	Y <—>	Open	Character	Different	Yes
VAREITH	CLOB	Y <—>	Either	Character	Different	Yes
VARONLY	CLOB	Y <—>	Only	Character	Different	Yes
VARGRPH	CLOB	N	Graphic	Character	Different	—
VARUCS2	CLOB	Y <—>	UCS2	Character	Different	Yes
VARCHAR	CLOB	Y <—>	Character	Open	Different	Yes
VAROPEN	CLOB	Y <—>	Open	Open	Same/Different	No/Yes
VAREITH	CLOB	Y <—>	Either	Open	Different	Yes
VARONLY	CLOB	Y <—>	Only	Open	Different	Yes
VARGRPH	CLOB	Y <—>	Graphic	Open	Different	Yes
VARUCS2	CLOB	Y <—>	UCS2	Open	Different	Yes
VARCHAR	DBCLOB	N	Character	Graphic	Different	–
VAROPEN	DBCLOB	Y <—>	Open	Graphic	Different	Yes
VAREITH	DBCLOB	Y <—>	Either	Graphic	Different	Yes
VARONLY	DBCLOB	Y <—>	Only	Graphic	Different	Yes
VARGRPH	DBCLOB	Y <—>	Graphic	Graphic	Same/Different	No/Yes
VARUCS2	DBCLOB	Y <—>	UCS2	Graphic	Different	Yes
VARCHAR	DBCLOB	Y <—>	¬65535	UCS2	Different	Yes
VAROPEN	DBCLOB	Y <—>	¬65535	UCS2	Different	Yes
VAREITH	DBCLOB	Y <—>	¬65535	UCS2	Different	Yes
VARONLY	DBCLOB	Y <—>	¬65535	UCS2	Different	Yes
VARGRPH	DBCLOB	Y <—>	Graphic	UCS2	Different	Yes
VARUCS2	DBCLOB	Y <—>	UCS2	UCS2	Same/Different	No/Yes

Table 15. From-file and to-file mapping between variable-length data types and large objects (continued)

Field A type	Field B type	Allowed and copy direction	Data CCSID or attributes		CCSIDs	Conversion translation performed
			Field A	Field B		
VARCHAR	DBCLOB	N	65535	UCS2	Different	–
VAROPEN	DBCLOB	N	65535	UCS2	Different	–
VAREITH	DBCLOB	N	65535	UCS2	Different	–
VARONLY	DBCLOB	N	65535	UCS2	Different	–

Copy between different systems

You can use the following commands to import (load) or export (unload) data to and from AS/400:

- The **Copy From Import File (CPYFRMIMPF) command** maps or parses the data from (″import″) an import file to the to-file.

  For more information on the CPYFRMIMPF command, see "Notes on the CPYFRMIMPF command" and "Restrictions on the CPYFRMIMPF command" on page 123. Depending on what type of file the import file is, there are different steps to use when running CPYFRMIMPF. See the following topics for more information on the appropriate steps:

  – "(CPYFRMIMPF) Importing data to the AS/400 when the from-file is a database file or DDM file" on page 124

  – "(CPYFRMIMPF) Importing data to AS/400 when the import file is a stream file" on page 125

  The CPYFRMIMPF command also supports a parallel data loader to copy information from an import file to a to-file using multiple jobs during the copy. To use multiple jobs, the system must have the Symmetric Multiprocessing Product (SMP). See "Parallel data loader support to use with the CPYFRMIMPF command" on page 125 for more information on Parallel Data Loader support.

- The **Copy To Import File (CPYTOIMPF) command** copies the data from the from-file to an import file. You can then move the import file (or file to be exported) to your platform by any method you choose. Your system then handles the data from the import file in one of two ways. See "Handling data from the import file" on page 125 for more information.

  The user can also specify a stream file, and the CPYTOIMPF will copy the data to the stream file. For more information on the CPYTOIMPF command, see "Notes on the CPYTOIMPF command" on page 129.

Notes on the CPYFRMIMPF command

The authority needed to perform the copy is similar to the authority requirements for all other copies.

The from-file can be any of the following:

- A stream file
- A DDM file
- A tape file
- A source physical file

- A distributed physical file
- A program described physical file
- A single format logical file
- An externally described physical file with one field. The one field cannot be a numeric data type.

The to-file can be any of these:

- A source file
- A DDM file
- A distributed physical file
- A program described physical file
- An externally described physical file

The field definition file can be any of these:

- A source physical file
- A DDM file
- A program described physical file
- An externally described physical file with one field

The error file can be any of the following:

- A source physical file
- A DDM file
- A program described physical file
- An externally described physical file with one field

Note: The format of the error file and from-file must be the same.

Restrictions on the CPYFRMIMPF command

The following restrictions apply to the CPYFRMIMPF command:

- The data type of the from-file can be one of two types:
  - A source physical file
  - A physical file with one field with a data type of CHARACTER, IGC OPEN, IGC EITHER, IGC ONLY, GRAPHIC, or variable length
- The copied records do not have the same relative record numbers in the to-file as in the from-file.
- Create the to-file prior to copying.
- The command restricts the correct usage for delimiters.
- The to-file and from-file cannot be the same file.
  - If a record from the from-file cannot be imported, the process continues based on the Errors Allowed (ERRLVL) parameter.
  - If the from-file is a stream file, a temporary database file is created in QRECOVERY. The naming convention for these types of files is QACPXXXXXX where the system fills in XXXXXX.
  - If the from-file is a source file, the system does not copy the first 12 bytes of the record (Sequence field and Date field). If the to-file is a source file, the system sets the first 12 bytes of the to-file's data (Sequence field and Date field) to zeros.

You can use this command on files that contain user-defined types (UDTs) and user-defined functions (UDFs). You cannot use this command on files that contain large objects (LOBs) or DataLinks (DLs).

(CPYFRMIMPF) Importing data to the AS/400 when the from-file is a database file or DDM file

The from-file contains the data you want to import to AS/400. To import data for a database file or DDM file, follow these steps:

1. Create an import file for the data that you can copy to a DB2 for AS/400 externally described file. The import file can be a database source file, an externally described database file which has one field, or a program described physical file. If the file has one field, the data type must be CHARACTER, IGC OPEN, IGC EITHER, IGC ONLY, GRAPHIC, or variable length. The record length of the import file should be long enough to contain the longest record of the file being sent to the AS/400.
2. Send the data to the import file or from-file. Sending the data into the import file causes the necessary ASCII to EBCDIC data conversions to occur. There are several ways to import the data such as:
   - TCP/IP file transfer (text transfer)
   - CA/400 support (file transfer, ODBC)
   - CPYFRMTAP command (copy from tape file)
3. Create a DB2 for AS/400 externally described database file, or DDM file, which will contain the resultant data of the import file.
4. Use the CPYFRMIMPF command to copy (translate or parse the records) from the import file to the to-file. For importing large files, you can choose to have the import file split-up into multiple parts so that each part can be processed in parallel on an N-way multi-processor system. See “Parallel data loader support to use with the CPYFRMIMPF command” on page 125 for more information about using multiple jobs during the copy.
5. You should also use the following “Tips to improve copy performance”.

Tips to improve copy performance

Follow these steps to inprove the performance of the CPYFRMIMPF command:

1. Delete any logical keyed files based on the TOFILE.
2. Remove all constraints and triggers of the TOFILE.
3. Ensure the FROMFILE records will be copied correctly by attempting to copy a few of the records. Copy a few of the records using the FROMRCD and number of records option.
4. Use the ERRLVL(*NOMAX) parameter after you know you can copy the data correctly.
5. When the ERRLVL(*NOMAX) parameter is used, record blocking increases performance. If an error in writing a record occurs during record blocking, the number of records listed as being copied in the completion message, CPC2955, may not be accurate.

(CPYFRMIMPF) Importing data to AS/400 when the import file is a stream file

If the import file is a stream file, use the following steps for importing data to the AS/400:

1. Create a DB2 for AS/400 externally described database file, or DDM file, which will contain the resultant data of the import file.
2. Use the CPYFRMIMPF command to copy (translate or parse the records) from the import file to the to-file. For importing large files, you can have the import file split-up into multiple parts. The multiple parts then process in parallel.
3. When the stream file (the import file) has records to copy to the externally described database or DDM file (the to-file), a temporary file is created to contain the job or jobs from that stream file. The temporary file, created by the system, acts as an intermediate place for holding records, and then copying them on to the to-file. The system then deletes the temporary file when the copy function from the temporary file to the database or DDM files completes.

Parallel data loader support to use with the CPYFRMIMPF command

The Copy From Import File (CPYFRMIMPF) supports copying the data in parallel from an import file to a to-file using multiple jobs during the copy. This allows you to copy data files from other platforms into a to-file quickly and easily. This is especially useful for those who use data warehousing. To use multiple jobs, your system must have the Symmetric Multiprocessing Product (SMP).

The number of jobs you use during the copy is determined by the DEGREE(*NBRTASKS) parameter of the Change Query Attributes (CHGQRYA) command. If the from-file has less than 50,000 records, only one job will be used regardless of the *NBRTASKS value.

The CPYFRMIMPF command (with the parallel data loader support) essentially breaks the import file into smaller portions or blocks. Each of these smaller portions is submitted in parallel, so the entire file processes at the same time. (This eliminates the latency of sequential processing.)

To maintain the same relative record numbers of the from-file in the to-file, use only one job for the copy. Specify DEGREE(*NONE).

Handling data from the import file

The Copy From Import File (CPYFRMIMPF) reads data from an import file and copies the data to a to-file. The data of the import file can be formatted by delimiters or in a fixed format.

A “Delimited Import File” on page 126 has a series of characters (delimiters) that define where fields begin and end. The parameters of the command define which characters are used for delimiters.

A “Fixed Formatted Import File” on page 128 requires the user to define a Field Definition File that defines the format of the import file. The Field Definition File defines where fields begin, end, and are null.

Delimited Import File

The following characters and data types interpret the import file's data for a delimited import file:

Blanks

Blanks are treated in the following ways:

- All leading and trailing blanks are discarded for character fields unless enclosed by string delimiters.
- A field of all blanks is interpreted as a null field for character data.
- You cannot embed blanks within numeric fields.
- You cannot select a blank as a delimiter.

Null Fields

A null field is defined as:

- Two adjacent field delimiters (no data in between).
- A field delimiter followed by a record delimiter (no data in between), an empty string.
- A field of all blanks

If the field is null, the following is true:

- If the record's output field is not nullable and the import is a null field, the record is not to be copied, and an error is signaled.

Delimiters

- A delimiter cannot be a blank
- A string delimiter cannot be the same as a field delimiter, record delimiter, date separator, or time separator.
- A string delimiter can enclose all non-numeric fields (character, date, time, and so forth). The string delimiter character should not be contained within the character string.
- A field delimiter and a record delimiter can be the same character.
- The defaults for delimiters are as follows:
  - String is: ″ Double quote.
  - Field is: , Comma.
  - Decimal point is: . Period.
  - Record is *EOR End of record.
- If the data type of the from is CHARACTER, OPEN, EITHER, or ONLY, all double byte data must be contained within string delimiters or shift characters (for OPEN, EITHER, ONLY data type).

Numeric Field

- Numeric fields can be imported in decimal or exponential form.
- Data to the right of the decimal point may be truncated depending on the output data format.
- Decimal points are either a period or a comma (command option).
- Signed numeric fields are supported, + or -.

Character or Varcharacter Fields

- Fields too large to fit in the output fields are truncated (right). The system sends a diagnostic message.

- An empty string is defined as two string delimiters with no data between them.
- For the system to recognize a character as a starting string delimiter, it must be the first non-blank character in the field. For example, 'abc' with ' as the delimiter is the same as abc.
- Data after an ending string delimiter and before a field or record delimiter is discarded.

IGC or VarIGC Fields

- The system copies data from the from-file to the to-file. If any of the data is not valid, the system generates a mapping error.
- Data located between the Shift Out and Shift In characters is treated as double-byte data. This data is also not parsed for delimiters. The Shift characters in this case become ″string delimiters″.

Graphic, VarGraphic Fields

The system copies the data from the from-file to the to-file.

CCSIDs

- The data from the from-file is read into a buffer by the CCSID of the from-file. The data in the buffer is checked and written to the to-file. The CCSID of the open to-file is set to the value of the from-file, unless a to-file CCSID is used. If a to-file CCSID is used, the data is converted to that CCSID. If the from-file is a tape file, and the FROMCCSID(*FILE) is specified, the following limits apply:
  - The job CCSID is used or
  - The from-file CCSID is requested by the user
- The character data (delimiters) passed on the command are converted to the CCSID of the from file. This allows the character data of the from-file and command parameters to be compatible.

Date Field

- All date formats supported by AS/400 can be imported (*ISO, *USA, *EUR, *JIS, *MDY, *DMY, *YMD, *JUL, and *YYMD).
- You can copy a date field to a timestamp field.

Time Field

- All time formats supported by AS/400 can be imported (*ISO, *USA, *EUR, *JIS, *HMS).
- You can copy a time field to a timestamp field.

Date and Time Separators

The system supports all valid separators for date and time fields.

Timestamp Field

Timestamp import fields must be 26 bytes. The import ensures that periods exist in the time portion, and a dash exists between the date and time portions of the timestamp.

Number of Fields Mismatch

If the from-file or to-file do not have the same number of fields, the data is either truncated to the smaller to-file size, or the extra to-file fields will receive a null value. If the fields cannot contain null values, the system issues an error message.

Multiple Jobs

The number of jobs that are used to copy the data depends on the

DEGREE(*NBRTASKS) parameter of the CHGQRYA command. When multiple jobs are used, the system uses batch jobs to copy the data. The user can change, hold or end these batch jobs. The copy does not complete until all the started batch jobs complete.

The relative record numbers can be maintained only if a single job is used and the import file does not contain any deleted records. If the from-file is a distributed physical file or logical file, the system performs the copy in a single process.

Files with less than 50,000 records use only one job.

Fixed Formatted Import File

Below is an example of a Field Definition File that describes the fixed formatted file:

```
 -*******************************************************/
 -****            Field Definition File                  */
 -*******************************************************/
- Description: This Field Definition File
- defines the import's file                              */
-       (FROMFILE) field start and end positions.        */
-*******************************************************/
-       (FROMFILE) field start and end positions.        */
-FILE MYLIB/MYFILE                                       */
field1   1    12   13
field2  14    24    0
field3  25    55   56
field4  78    89   90
field5 100   109    0
field6 110   119  120
field7 121   221    0
*END


The following is a brief explanation of the
Field Definition File format:

     -     =  Comment line
     *END  =  End of definition, this must be included

Field     Starting     Ending         Null
Name      Position     Position       Character Position
_____________________________________________________

 field1   1            12             13
 field2   14           24             None
 field3   25           55             56
 field4   78           89             90
 field5  100          109             None
 field6  110          119             120
 field7  121          221             None
```

Field Name

This name is the name of the to-file field name.

Starting Position

This is the starting position for the field in the import file of each record. This is the byte position.

Ending Position

This is the ending position for the field in the import file of each record. This is the byte position.

Null Character Position

This is the position for the NULL value for the field in the import file of each record. The value zero specifies that NULL does not have a value. The value in the import file can be 'Y' or 'N'.

'Y' means the field is NULL. 'N' means the field is not NULL.

Each column must be separated by a blank character.

Notes on the CPYTOIMPF command

The Copy To Import File (CPYTOIMPF) command reads data from a user from-file and copies it into an import file. The number of jobs used for copy is one. The data of the import file can be formatted by delimiters or it can be in a fixed format. A "Notes on the delimited import file (CPYTOIMPF command)" has a series of characters (delimiters) that are used to define where fields begin and end. See "Restrictions for the CPYTOIMPF command" on page 130 for more information.

The parameters of the command define which characters are used for delimiters. A fixed format import file uses a fixed format. For more information on this, see "Copying data to the import file in a fixed format (CPYTOIMPF command)" on page 131.

The data in the from-file is read from the formatted database file and written to the import file with the parameters from the command.

The from-file can be any of these:

- A source physical file
- A program described physical file
- A distributed physical file
- A single format logical file
- An externally described physical file

The to-file can be any of these:

- a stream file
- a source physical file
- a program described physical file
- a distributed physical file with one non-numeric field
- an externally described physical file with one non-numeric field

Notes on the delimited import file (CPYTOIMPF command)

Null Fields

If a field is null, the field contains two adjacent field delimiters (no data in between).

Delimiters

- A delimiter cannot be a blank.
- A period cannot be a character string delimiter.
- A string delimiter cannot be the same as a field or record delimiter.
- A field and record delimiter can be the same character.
- The defaults for delimiters are as follows:
  - String is: ″ Double quote.

- Field is: , Comma.
- Decimal point is: . Period.
- Record is: *EOR End of record.

Numeric Fields
Decimal points are either a period or a comma (command option).

Graphic Fields
The string delimiter is placed around all graphic data. If graphic data is contained in the file and the string delimiter is the *NONE value, an error is signaled.

All Fields
The CAST function in SQL copies the data from the from-file to the to-file. All data is copied and the relative record numbers of the from-file and to-file are the same, unless the from-file contains deleted records. Deleted records are not copied.

CCSIDs
The data from the from-file is read into the to-file's CCSID.

Date Fields
All date formats supported by AS/400 can be exported (*ISO, *USA, *EUR, *JIS, *MDY, *DMY, *YMD, *JUL, *YYMD).

Time Fields
All time formats supported by AS/400 can be exported (*ISO, *USA, *EUR, *JIS, *HMS).

Date and Time Separators
All valid separators are supported for date and time fields.

Timestamp Fields
Timestamp export fields must be 26 bytes.

Restrictions for the CPYTOIMPF command

The following restrictions apply to the CPYTOIMPF command:

- The command restricts the correct usage for delimiters.
- The data type of a database file for the to-file can be any of the following:
  - CHARACTER, IGC OPEN, IGC EITHER, IGC ONLY, GRAPHIC, or variable length. Its length must be capable of containing the data of the from-file, separators, and any data conversions.
  - The to-file and from-file cannot be the same file.
  - The from-file cannot be a multi-formatted logical file.
- If the to-file's record length is not long enough, an error is signaled.

You can use this command on files that contain user-defined types (UDTs) and user-defined functions (UDFs). You cannot use this command on files that contain large objects (LOBs) or DataLinks (DLs).

Copying data to the import file in a fixed format (CPYTOIMPF command)

When you copy data to the import file in a fixed format (DTAFMT(*FIXED)), each field of the file is copied. A null indicator on the command NULLS(*YES) places either a 'Y' or 'N' following the field data in the to-file indicating if the field is null or not.

Chapter 5. Working with spooled files

Spooling functions help system users to manage input and output operations more efficiently. The system supports two types of spooling:

- "Output spooling" sends job output to disk storage, rather than directly to a printer or diskette output device. Output spooling allows the job that produces the output to continue processing without consideration for the speed or availability of output devices.
- "Input spooling" on page 143 accepts job input, stores the input data in disk storage, and allows the input device to be used independently of when the job is actually processed.

Output spooling may be used for both printer and diskette devices; input spooling applies to diskette and database file input.

This chapter discusses both output and input spooling, including advanced output spooling support, such as using multiple output queues and redirecting files. For more information about spooling support for printer and diskette devices, see the *Printer Device Programming* book and the *Tape and Diskette Device Programming* book, respectively.

Output spooling

Output spooling allows the system to produce output on multiple output devices, such as printer and diskette devices, in an efficient manner. It does this by sending the output of a job destined for a printer or diskette to disk storage. This process breaks a potential job limitation imposed by the availability or speed of the output devices.

Spooling is especially important in a multiple-user environment where the number of jobs running often exceeds the number of available output devices. Using output spooling, the output can be easily redirected from one device to another.

The main elements of output spooling are:

Device description
A description of the printer or diskette device

Spooled file
A file containing spooled output records that are to be processed on an output device

Output queue
An ordered list of spooled files

Writer A program that sends files from an output queue to a device

Application program
A high-level language program that creates a spooled file using a device file with the spooling attribute specified as SPOOL(*YES)

Device file
A description of the format of the output, and a list of attributes that describe how the system should process the spooled file

Figure 14 shows the relationship of these spooling elements.

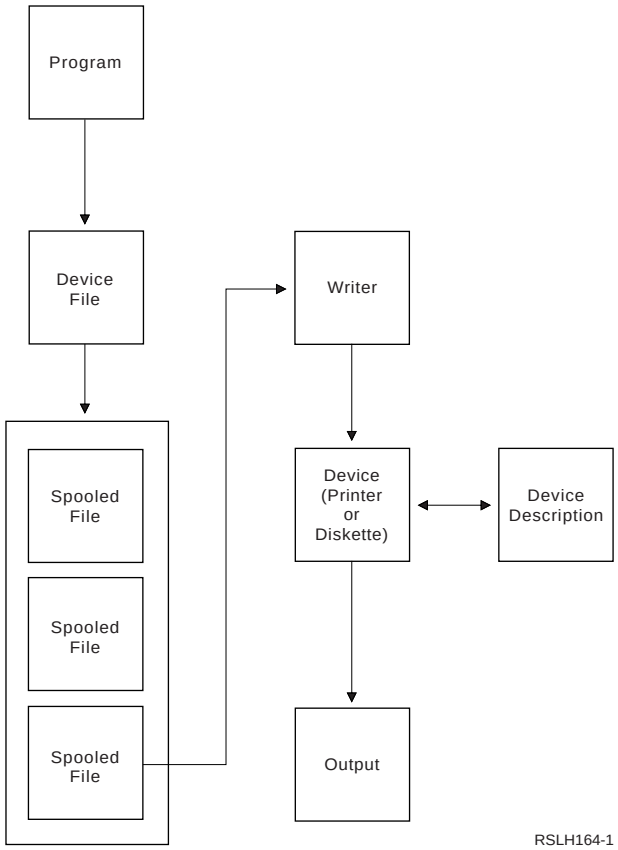


Figure 14. Relationship of Output Spooling Elements

Output spooling functions are performed by the system without requiring any special operations by the program that produces the output. When a device file is opened by a program, the operating system determines whether the output is to be spooled. When a printer or diskette device file specifying spooling is opened, the spooled file containing the output of the program is placed on the appropriate output queue in the system.

A spooled file can be made available for printing when the printer file is opened, when the printer file is closed, or at the end of the job. A printer writer is started in the spooling subsystem to send the records to the printer. The spooled file is selected from an output queue. The same general description applies for spooled diskette files.

Device Descriptions

Device descriptions must be created for each printer and diskette device to define that device to the system. Printer device descriptions are created using the Create Device Description for Printer (CRTDEVPRT) command; diskette device descriptions are created using the Create Device Description for Diskette (CRTDEVDKT) command. See the *Local Device Configuration* book for more information about specifying device descriptions.

Summary of Spooled File Commands

The following commands may be used to work with spooled files. For detailed descriptions of the commands, see the *CL Reference (Abridged)*.

CHGSPLFA
Change Spooled File Attributes: Allows you to change some attributes of a spooled file, such as the output queue name or the number of copies requested, while the spooled file is on an output queue.

CPYSPLF
Copy Spooled File: Copies a spooled file to a specified database file. The database file may then be used for other applications, such as those using microfiche or data communications.

DLTSPLF
Delete Spooled File: Deletes a spooled file from an output queue.

DSPSPLF
Display Spooled File: Allows you to display data records of a spooled file.

HLDSPLF
Hold Spooled File: Stops the processing of a spooled file by a spooling writer. The next spooled file in line will be processed.

RLSSPLF
Release Spooled File: Releases a previously held spooled file for processing by a spooling writer.

SNDNETSPLF
Send Network Spooled File: Sends a spooled file to another system user on the Systems Network Architecture distribution services (SNADS) network.

SNDTCPSPLF
Send TCP/IP Spooled File: Sends a spooled file to another system user using TCP/IP.

WRKSPLF
Work with Spooled Files: Allows you to display or print a list of spooled files on the system.

WRKSPLFA
Work with Spooled File Attributes: Shows the current attributes of a spooled file.

Locating Your Spooled Files

The Work with Spooled Files (WRKSPLF) command can be used to display or print all the spooled files that you have created. This is the easiest way to find your spooled files if you do not know the name of the output queue where they have been placed. To find all spooled files created by your current job, use the Work with Job (WRKJOB) command and choose Option 4 to work with the spooled files.

File Redirection

File redirection occurs when a spooled file is sent to an output device other than the one for which it was originally intended. File redirection may involve devices that process different media (such as printer output sent to a diskette device) or devices that process the same type of media but are of different device types (such as 5219 Printer output sent to a 4224 Printer).

Depending on the new output device for the spooled file, the file may be processed just as it would have been on the originally specified device. However, differences in devices often cause the output to be formatted differently. In these cases, the system sends an inquiry message to the writer's message queue to inform you of the situation and allow you to specify whether you want printing to continue. For more information about print file redirection, see the *Printer Device Programming* book.

Output Queues

Batch and interactive job processing may result in spooled output records that are to be processed on an output device, such as a printer or diskette drive. These output records are stored in spooled files until they can be processed. There may be many spooled files for a single job.

When a spooled file is created, the file is placed on an output queue. Each output queue contains an ordered list of spooled files. A job can have spooled files on one or more output queues. All spooled files on a particular output queue should have a common set of output attributes, such as device, form type, and lines per inch. Using common attributes on an output queue reduces the amount of intervention required and increases the device throughput.

The following lists the parameters on the Create Output Queue (CRTOUTQ) command and what they specify:

- MAXPAGES: Specifies the maximum spooled file size in pages that is allowed to be printed between a starting and ending time of day.
- AUTOSTRWTR: Specifies the number of writers that are started automatically to this output queue.
- DSPDTA: Whether users without any special authority but who do have *USE authority to the output queue can display, copy, or send the contents of spooled files other than their own. By specifying *OWNER for DSPDTA, only the owner of the file or user with *SPLCTL can display, copy, or send a file.
- JOBSEP: How many, if any, job separator pages are to be printed between the output of each job when the output is printed.
- DTAQ: The data queue associated with this output queue. If specified, an entry is sent to the data queue whenever a spooled file goes to Ready Status on the queue.
- OPRCTL: Whether a user having job control authority can control the output queue (for example, if the user can hold the output queue).
- SEQ: Controls the order in which spooled files will be sorted on the output queue. See "Order of Spooled Files on an Output Queue" on page 138 for more information.
- AUTCHK: Specifies what type of authority to the output queue will enable a user to control the spooled files on the output queue (for example, enable the user to hold the spooled files on the output queue).
- AUT: Public authority. Specifies what control users have over the output queue itself.
- TEXT: Text description. Up to 50 characters of text that describes the output queue.

For a complete list of parameters for the CRTOUTQ command, see the *CL Reference (Abridged)*.

Summary of Output Queue Commands

The following commands may be used to create and control output queues. For detailed descriptions of the commands, see the *CL Reference (Abridged)*.

CHGOUTQ
Change Output Queue: Allows you to change certain attributes of an output queue, such as the sequence of the spooled files on the output queue.

CLROUTQ
Clear Output Queue: Removes all spooled files from an output queue.

CRTOUTQ
Create Output Queue: Allows you to create a new output queue.

DLTOUTQ
Delete Output Queue: Deletes an output queue from the system.

HLDOUTQ
Hold Output Queue: Prevents all spooled files on a particular output queue from being processed by a spooling writer.

RLSOUTQ
Release Output Queue: Releases a previously held output queue for processing by a spooling writer.

WRKOUTQ
Work with Output Queue: Shows the overall status of all output queues, or the detailed status of a specific output queue and its spooled files.

WRKOUTQD
Work with Output Queue Description: Shows descriptive information for an output queue.

Default Printer Output Queues

When a printer is configured to the system, the system automatically creates the printer's default output queue in library QUSRSYS. The output queue is given a text description of `'Default output queue for printer xxxxxxxxxx'`, where `xxxxxxxxxx` is the name of the printer.

The AUT parameter for the output queue is assigned the same value as that specified by the AUT parameter for the printer device description. All other parameters are assigned their default values. Use the Change Command Default (CHGCMDDFT) command to change the default values used when creating output queues with the CRTOUTQ command.

The default output queue for a printer is owned by the user who created the printer device description. In the case of automatic configuration, both the printer and the output queue are owned by the system profile QPGMR.

Default System Output Queues

The system is shipped with the defaults on commands to use the default output queue for the system printer as the default output queue for all spooled output. The system printer is defined by the QPRTDEV system value.

When a spooled file is created by opening a device file and the output queue specified for the file cannot be found, the system will attempt to place the spooled

file on output queue QPRINT in library QGPL. If for any reason the spooled file cannot be placed on output queue QPRINT, an error message will be sent and the output will not be spooled.

The following output queues are supplied with the system:

QDKT Default diskette output queue

QPRINT
Default printer output queue

QPRINTS
Printer output queue for special forms

QPRINT2
Printer output queue for 2-part paper

Creating Your Own Output Queues

You can create output queues for each user of the system. For example:

```
CRTOUTQ OUTQ(QGPL/JONES) +
  TEXT('Output queue for Mike Jones')
```

Order of Spooled Files on an Output Queue

The order of spooled files on an output queue is mainly determined by the status of the spooled file. A spooled file that is being processed by a writer may have one of the following statuses:

Status	Description
PRT	Printing
WTR	Writer
PND	Pending to be printed
SND	Sending to another system

Spooled files with these statuses are placed at the top of the output queue. A spooled file being processed by the writer may have a held (HLD) status if a user has held the spooled file, but the writer is not yet finished processing the file. All other files with a status of RDY are listed on the output queue after the file being processed by a writer, followed by files with statuses other than RDY.

Within each type of spooled file (RDY and non-RDY files) the following information causes a further ordering of the files. The items are listed in sequence based on the amount of importance they have on the ordering of spooled files, with the first item having the most importance.

1. The output priority of the spooled file.
2. A date and time field (time stamp).

   For output queues with SEQ(*JOBNBR) specified, the date and time that the job which created the spooled file entered the system are the date and time field. (A sequential job number is also assigned to the job when it enters the system.)

   For output queues with SEQ(*FIFO) specified, the date and time field is set to the current system date and time when any of the following occur:

   - A spooled file is created by opening a device file.
   - The output priority of the job which created the spooled file is changed.

- The status of the spooled file changes from RDY to HLD, SAV, OPN, or CLO; or the status changes from HLD, SAV, OPN, or CLO to RDY.
- A spooled file is moved to another output queue which has SEQ(*FIFO) specified.

3. The SCHEDULE parameter value of the spooled file.

   Files with SCHEDULE(*JOBEND) specified are grouped together and placed after other spooled files of the same job that have SCHEDULE(*IMMED) or SCHEDULE(*FILEEND) specified.
4. The spool number of the file.

Because of the automatic sorting of spooled files, different results occur when SEQ(*JOBNBR) is specified for an output queue than when SEQ(*FIFO) is specified. For example, when a spooled file is held and then immediately released on an output queue with SEQ(*JOBNBR) specified, the file will end up where it started; but if the same file were held and then immediately released on an output queue with SEQ(*FIFO) specified, the file would be placed at the end of the spooled files which have the same priority and a status of RDY.

Using Multiple Output Queues

You may want to create multiple output queues for:

- Special forms printing
- Output to be printed after normal working hours
- Output that is not printed

  An output queue can be created to handle spooled files that need only to be displayed or copied to a database file. Care should be taken to remove unneeded spooled files.
- Special uses

  For example, each programmer could be given a separate output queue.
- Output of special IBM files

  You may want to consider separate queues for the following IBM-supplied files:
  - QPJOBLOG: You may want all job logs sent to a separate queue.
  - QPPGMDMP: You may want all program dumps sent to a separate queue so you can review and print them if needed or clear them daily.
  - QPSRVDMP: You may want all service dumps sent to a separate queue so the service representative can review them if needed.

Output Queue Recovery

If a job that has produced spooled files is running when the job or system stops abnormally, the files remain on the output queue. Some number of records written by active programs may still be in main storage when the job ends and will be lost. You should check these spooled files to ensure that they are complete before you decide to continue using the files.

You can use the SPLFILE parameter on the End Job (ENDJOB) command to specify if all spooled files (except QPJOBLOG) created by the job are to be kept for normal processing by the writer, or if these files are to be deleted.

If an abnormal end occurs, the spooled file QPJOBLOG will be written at the next IPL of the system.

If a writer fails while a spooled file is being printed, the spooled file remains on the output queue intact.

If an output queue becomes damaged such that it cannot be used, you will be notified by a message sent to the system operator message queue. The message will come from a system function when a writer or a job tries to put or take spooled files from the damaged queue.

A damaged output queue can be deleted using the Delete Output Queue (DLTOUTQ) command, or it will be deleted by the system during the next IPL. After a damaged output queue is deleted, all spooled files on the damaged output queue are moved to output queue QSPRCLOUTQ in library QRCL. This is done by the QSPLMAINT system job, which issues completion message CPC3308 to the QSYSOPR message queue when all spooled files have been moved to the QSPRCLOUTQ output queue.

After the damaged output queue is deleted, it can be created again by entering the Create Output Queue (CRTOUTQ) command. Spooled files on output queue QSPRCLOUTQ can be moved back to the newly created output queue using the Change Spooled File Attributes (CHGSPLFA) command.

Note: If the output queue that was damaged was the default output associated with a printer, the system will automatically re-create the output queue when it is deleted. This system-created output queue will have the same public authority as specified for the device and default values for the other parameters. After the system re-creates the output queue, you should verify its attributes are correct and change them as needed. The output queue can be changed using the Change Output Queue (CHGOUTQ) command. When a damaged output queue associated with a printer is deleted and created again, all spooled files on the damaged queue will be moved to the re-created output queue. This is done by the QSPLMAINT system job, which issues completion message CPC3308 to the QSYSOPR message queue when all spooled files have been moved.

Spooling Writers

A writer is an OS/400 program that takes spooled files from an output queue and produces them on an output device. The spooled files that have been placed on a particular output queue will remain stored in the system until a writer is started to the output queue.

The writer takes spooled files one at a time from the output queue, based on their priority. The writer processes a spooled file only if its entry on the output queue indicates that it has a ready (RDY) status. You can display the status of a particular spooled file using the Work with Output Queue (WRKOUTQ) command.

If the spooled file has a ready status, the writer takes the entry from the output queue and prints the specified job and/or file separators, followed by the output data in the file. If the spooled file does not have a ready status, the writer leaves the entry on the output queue and goes on to the next entry. In most cases the writer will continue to process spooled files (preceded by job and file separators) until all files with a ready status have been taken from the output queue.

The AUTOEND parameter on the start writer commands determines whether the writer continues to wait for new spooled files to become available to be written, end after processing one file, or end after all spooled files with ready status have been taken from the output queue.

Summary of Spooling Writer Commands

The following commands may be used to control spooling writers. For detailed descriptions of the commands, see the *CL Reference (Abridged)*.

STRDKTWTR
Start Diskette Writer: Starts a spooling writer to a specified diskette device to process spooled files on that device.

STRPRTWTR
Start Printer Writer: Starts a spooling writer to a specified printer device to process spooled files on that device.

STRRMTWTR
Start Remote Writer: Starts a spooling writer that sends spooled files from an output queue to a remote system.

CHGWTR
Change Writer: Allows you to change some writer attributes, such as form type, number of file separator pages, or output queue attributes.

HLDWTR
Hold Writer: Stops a writer at the end of a record, at the end of a spooled file, or at the end of a page.

RLSWTR
Release Writer: Releases a previously held writer for additional processing.

ENDWTR
End Writer: Ends a spooling writer and makes the associated output device available to the system.

Spooled File Security

Spooled file security is primarily controlled through the output queue which contains the spooled file. In general, there are four ways that a user can become authorized to control a spooled file (for example, hold or release the spooled file):

- User is assigned spool control authority (SPCAUT(*SPLCTL)) in the user's user profile.
- User is assigned job control authority (SPCAUT(*JOBCTL)) in the user's user profile and the output queue is operator controllable (OPRCTL(*YES)).
- User has the required object authority for the output queue. The required object authority is specified by the AUTCHK keyword on the CRTOUTQ command. A value of *OWNER indicates that only the owner of the output queue is authorized via the object authority for the output queue. A value of *DTAAUT indicates that users with *CHANGE authority to the output queue are authorized to control the output queue.

  Note: The specific authority required for *DTAAUT are *READ, *ADD, and *DLT data authorities.
- A user is always allowed to control the spooled files created by that user.

For the Copy Spooled File (CPYSPLF), Display Spooled File (DSPSPLF), and Send Network Spooled File (SNDNETSPLF) commands, in addition to the four ways already listed, there is an additional way a user can be authorized. If DSPDTA(*YES) was specified when the output queue was created, any user with *USE authority to the output queue will be allowed to run these commands. The specific authority required is *READ data authority. Copying, displaying, sending, and moving a file to another output queue by changing the spooled file can be limited by specifying DSPDTA(*OWNER). Then only the owner of the spooled file or user with *SPLCTL can perform these operations on the spooled file.

See the *CL Reference (Abridged)* for details about the authority requirements for individual commands.

To place a spooled file on an output queue, one of the following authorities is required:

- User is assigned spool control authority (SPCAUT(*SPLCTL)) in the user's user profile.
- User is assigned job control authority (SPCAUT(*JOBCTL)) in the user's user profile and the output queue is operator controllable (OPRCTL(*YES)).
- User has *READ authority to the output queue. This authority can be given to the public by specifying (AUT(*USE)) on the CRTOUTQ command.

Controlling the Number of Spooled Files in Your System

The number of spooled files in your system should be limited. When a job is completed, spooled files and internal job control information are kept until the spooled files are printed or canceled. The number of jobs on the system and the number of spooled files known to the system increase the amount of time needed to perform IPL and internal searches, and increases the amount of temporary storage required.

The number of jobs known to the system can be displayed using the Work with System Status (WRKSYSSTS) command.

You can use the Work with Spooled Files (WRKSPLF) command to identify spooled files that are no longer needed. By periodically entering the command:

```
WRKSPLF SELECT(*ALL)
```

you can determine which spooled files are older than 2 or 3 days, then delete the spooled files or contact the users who created them.

For detailed information on minimizing the number of job logs (for example, by using LOG(4 0 *NOLIST)), see the *CL Programming* book. For information regarding the use of system values to control the amount of storage associated with jobs and spooled files, refer to the *Work Management* book. To control the storage used on your system see "Spooling Library" on page 151.

Command Examples for Additional Spooling Support

You can define some functions to provide additional spooling support. Example source and documentation for the commands, files, and programs for these functions are part of library QUSRTOOL, which is an optionally installed part of the OS/400 program.

Input spooling

Input spooling takes the information from the input device, prepares the job for scheduling, and places an entry in a job queue. Using input spooling, you can usually shorten job run time, increase the number of jobs that can be run sequentially, and improve device throughput.

The main elements of input spooling are:

Job queue
An ordered list of batch jobs submitted to the system for running and from which batch jobs are selected to run.

Reader
A function that takes jobs from an input device or a database file and places them on a job queue.

When a batch job is read from an input source by a reader, the commands in the input stream are stored in the system as requests for the job, the inline data is spooled as inline data files, and an entry for the job is placed on a job queue. The job information remains stored in the system where it was placed by the reader until the job entry is selected from the job queue for processing by a subsystem. Figure 15 shows this relationship.

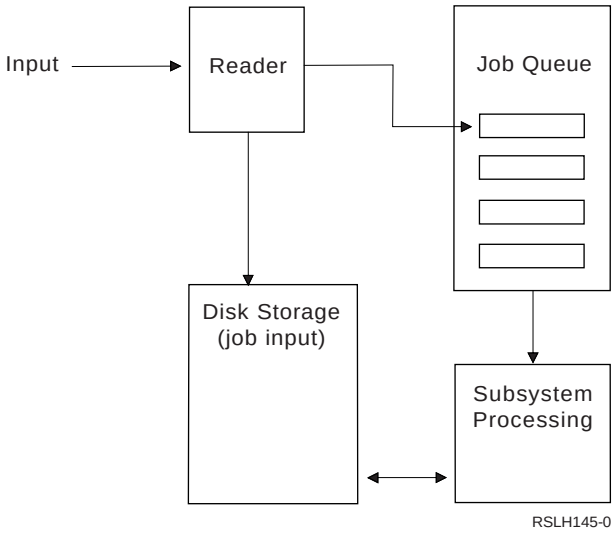


Figure 15. Relationship of Input Spooling Elements

You can use the reader functions to read an input stream from diskette or database files. Figure 16 on page 144 shows the typical organization of an input stream:

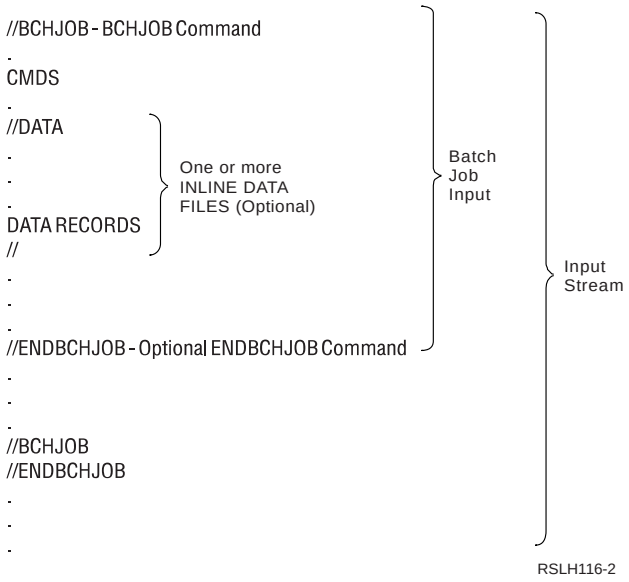


Figure 16. Typical Organization of an Input Stream

The job queue on which the job is placed is specified on the JOBQ parameter on the BCHJOB command, on the start reader command, or in the job description. If the JOBQ parameter on the BCHJOB command is:

- *RDR: The job queue is selected from the JOBQ parameter on the start reader command.
- *JOBD: The job queue is selected from the JOBQ parameter in the job description.
- A specific job queue: The specified queue is used.

For jobs with small input streams, you may improve system performance by not using input spooling. The submit job commands (SBMDBJOB and SBMDKTJOB) read the input stream and place the job on the job queue in the appropriate subsystem, bypassing the spooling subsystem and reader operations.

If your job requires a large input stream to be read, you should use input spooling (STRDKTRDR or STRDBRDR command) to allow the job to be input independent of when the job is actually processed.

Summary of Job Input Commands

The following commands may be used when submitting jobs to the system. The start reader commands may be used for spooling job input; the submit job commands do not use spooling. For detailed descriptions of these commands, see the *CL Reference (Abridged)*.

BCHJOB
Batch Job: Marks the start of a job in a batch input stream and defines the operating characteristics of the job.

DATA Data: Marks the start of an inline data file.

ENDBCHJOB
End Batch Job: Marks the end of a job in a batch input stream.

ENDINP
End Input: Marks the end of the batch input stream.

SBMDBJOB
Submit Database Jobs: Reads an input stream from a database file and places the jobs in the input stream on the appropriate job queues.

SBMDKTJOB
Submit Diskette Jobs: Reads an input stream from diskette and places the jobs in the input stream on the appropriate job queues.

STRDBRDR
Start Database Reader: Starts a reader to read an input stream from a database file and places the job in the input stream on the appropriate job queue.

STRDKTRDR
Start Diskette Reader: Starts a reader to read an input stream from diskette and places the job in the input stream on the appropriate job queue.

Job Queues

A job queue is an ordered list of jobs waiting to be processed by a particular subsystem. Jobs will not be selected from a job queue by a subsystem unless the subsystem is active and the job queue is not held. You can use job queues to control the order in which jobs are run.

A base set of job queues is provided with your system. In addition, you may create additional job queues that you need.

IBM-Supplied Job Queues

Several job queues are provided by IBM when your system is shipped. IBM supplies job queues for each IBM-supplied subsystem.

QCTL Controlling subsystem queue

QBASE
QBASE subsystem job queue

QBATCH
Batch subsystem queue

QINTER
Interactive subsystem queue

QPGMR
Programmer subsystem queue

QSPL Spooling subsystem queue

QSYSSBSD
QSYSSBSD subsystem job queue

QS36MRT
QS36MRT job queue

QS36EVOKE
QS36EVOKE job queue

QFNC Finance subsystem job queue

QSNADS
QSNADS subsystem job queue

Using Multiple Job Queues

In many cases, using QBATCH as the only job queue with the default of one active job will be adequate for your needs. If this is not adequate, you may want to have multiple job queues so that some job queues are active during normal working hours, some are for special purposes, and some are active after normal working hours. For example, you could designate different job queues for:

- Long-running jobs so you can control how many jobs are active at the same time.

  You may also want these jobs to use a lower priority than the other batch jobs.
- Overnight jobs that are inconvenient to run during normal working hours.

  For example, to run a Reorganize Physical File Member (RGZPFM) command on a large database file requires an exclusive lock on the file. This means that other users cannot access the file while this operation is taking place. Additionally, this operation could take a long time. It would be more efficient to place this job on a job queue for jobs which run during off-shift hours.
- High-priority jobs.

  You may want to have a job queue to which all high-priority work is sent. You could then ensure that this work is completed rapidly and is not delayed by lower-priority jobs.
- Jobs that are directed to particular resource requirement such as diskette or tape.

  Such a job queue would need a MAXACT parameter of 1 in the job queue entry of the subsystem description so that only one job at a time uses the resource.

  For example, if a tape is used for several jobs, all jobs using tape would be placed on a single job queue. One job at a time would then be selected from the job queue. This would ensure that no two jobs compete for the same device at the same time. If this happened, it would cause one of the jobs to end with an allocation error.

  Note: Tape output cannot be spooled.
- Programmer work.

  You may want to have a job queue to handle programmer work or types of work that could be held while production work is being run.
- Sequential running of a series of jobs.

  You may have an application in which one job is dependent on the completion of another job. If you place these jobs on a job queue that selects and runs one job at a time, this would ensure the running sequence of these jobs.

  If a job requires exclusive control of a file, you may want to place it on a job queue when the queue is the only one active on the system, such as during the night or on a weekend.

If you use multiple job queues, you will find that control of the various job queues is a main consideration. You will usually want to control:

- How many job queues exist.
- How many job queues are active in a particular subsystem at the same time.
- How many active jobs can be selected from a particular job queue at a particular time.
- How many jobs can be active in a subsystem at a particular time.

Creating Your Own Job Queues

There are numerous reasons why you may decide that you need job queues in addition to the ones supplied by IBM. Additional job queues can be created by using the Create Job Queue (CRTJOBQ) command:

```
CRTJOBQ QGPL/QNIGHT TEXT('Job queue for +
  night-time jobs')
```

The following lists the parameters on the Create Job Queue (CRTJOBQ) command and what they specify:

- OPRCTL: Specifies whether a user having job control authority can control the job queue (for example, if the user can hold the job queue).
- AUTCHK: Specifies what type of authority to the job queue will enable a user to control the jobs on the job queue (for example, enable the user to hold the jobs on the job queue).
- AUT: Specifies what control users have over the job queue itself.
- TEXT: Up to 50 characters of text that describe the job queue.

Multiple Job Queues for a Subsystem

If the priority and sequence of the next job queue to be used is important, you may want to assign and control multiple job queues for each subsystem. One use of multiple job queues is to establish a high-priority and a normal-priority job queue within a subsystem, allowing only one active job on each queue at any time.

Another example: If your production batch jobs need to be completed before a special after-hours job queue can be made active, you could have the last job in the normal batch job queue release the after-hours job queue.

Refer to the SEQNBR parameter in the Add Job Queue Entry (ADDJOBQE) command in the *CL Reference (Abridged)* to determine how to set priorities for jobs on job queues. For more information, refer to the *Work Management* book.

Using the WRKJOBQ Command

Jobs already on the job queue can be controlled using the Work with Job Queue (WRKJOBQ) command.

The WRKJOBQ command lists either:

- All the job queues on the system
- All the jobs on a specific job queue

The ability to list all the job queues is important when you are not sure what job queue was used for a job. From the list of all job queues, you can look at each job queue to find the job. The display of a specific job queue provides a list of all the jobs on the queue in the order in which they will become active.

Transferring Jobs

If a job is on a job queue and is not yet active, you can change the job to a different job queue using the JOBQ parameter on the Change Job (CHGJOB) command.

If a job becomes active, it is possible to place it back on a job queue. See the *Work Management* book for a discussion of the Transfer Job (TFRJOB) and Transfer Batch Job (TFRBCHJOB) commands.

Job Queue Security

You can maintain a level of security with your job queue by authorizing only certain persons (user profiles) to that job queue. In general, there are three ways that a user can become authorized to control a job queue (for example, hold or release the job queue):

- User is assigned spool control authority (SPCAUT(*SPLCTL)) in the user's user profile.
- User is assigned job control authority (SPCAUT(*JOBCTL)) in the user's user profile and the job queue can be controlled by the operator (OPRCTL(*YES)).
- User has the required object authority to the job queue. The required object authority is specified by the AUTCHK parameter on the CRTJOBQ command. A value of *OWNER indicates that only the owner of the job queue is authorized via the object authority for the job queue. A value of *DTAAUT indicates that users with *CHANGE authority for the job queue are authorized to control the job queue.

  Note: The specific authority required for *DTAAUT are *READ, *ADD, and *DLT data authority.

See the *CL Reference (Abridged)* for more information about authority requirements for individual commands.

These three methods of authorization apply only to the job queue, not to the jobs on the job queue. The normal authority rules for controlling jobs apply whether the job is on a job queue or whether it is currently running. See the *Work Management* book for details on the authority rules for jobs.

Job Queue Recovery

If a command fails or the system stops abnormally while a reader or a submit jobs command is running and a partial job (not all the input stream has been read) is placed on the queue, the entire job must be resubmitted to the job queue.

If a job is on a job queue when the system stops abnormally without damaging that job queue, the job remains intact on the job queue and is ready to run when the system becomes active again.

If the system stops abnormally while a job is running, the job is lost and must be resubmitted to the job queue.

If a job queue becomes damaged such that it cannot be used, you will be notified by a message sent to the system operator message queue. The message will come from a system function when a reader, Submit Jobs command, or a job tries to put or take jobs from the damaged queue.

A damaged job queue can be deleted using the Delete Job Queue (DLTJOBQ) command, or it will be deleted by the system during the next IPL. After a damaged job queue is deleted, all job files on the damaged job queue will be moved to output queue QSPRCLJOBQ in library QRCL. This is done by the QSPLMAINT system job, which issues completion message CPC3308 to the QSYSOPR message queue when all jobs have been moved to the QSPRCLJOBQ output queue.

After the damaged job queue is deleted, it can be created again by entering the Create Job Queue (CRTJOBQ) command. Jobs on the job queue QSPRCLOUTQ can be moved back to the newly created output queue using the Change Job (CHGJOB) command.

Using an Inline Data File

An inline data file is a data file that is included as part of a batch job when the job is read by a reader or a submit jobs command. An inline data file is delimited in the job by a //DATA command at the start of the file and by an end-of-data delimiter at the end of the file. The end-of-data delimiter can be a user-defined character string or the default of //.

The // must appear in positions 1 and 2. If your data contains a // in positions 1 and 2, you should use a unique set of characters such as:

```
// *** END OF DATA
```

To specify this as a unique end-of-data delimiter, the ENDCHAR parameter on the //DATA command should be coded as:

```
ENDCHAR('// *** END OF DATA')
```

Note: Inline data files can be accessed only during the first routing step of a batch job. If a batch job contains a Transfer Job (TFRJOB), a Reroute Job (RRTJOB), or a Transfer Batch Job (TFRBCHJOB) command, the inline data files cannot be accessed in the new routing step.

An inline data file can be either named or unnamed. For an unnamed inline data file, either QINLINE is specified as the file name in the //DATA command or no name is specified. For a named inline data file, a file name is specified.

A named inline data file has the following characteristics:

- It has a unique name in a job; no other inline data file can have the same name.
- It can be used more than once in a job.
- Each time it is opened, it is positioned to the first record.

To use a named inline data file, you must either specify the file name in the program or use an override command to change the file name specified in the program to the name of the inline data file. The file must be opened for input only.

An unnamed inline data file has the following characteristics:

- Its name is QINLINE. (In a batch job, all unnamed inline data files are given the same name.)
- It can only be used once in a job.
- When more than one unnamed inline data file is included in a job, the files must be in the input stream in the same order as when the files are opened.

To use an unnamed inline data file, do one of the following:

- Specify QINLINE in the program.
- Use an override file command to change the file name specified in the program to QINLINE.

If your high-level language requires unique file names within one program, you can use QINLINE as a file name only once. If you need to use more than one unnamed

inline data file, you can use an override file command in the program to specify QINLINE for additional unnamed inline data files.

Note: If you run commands conditionally and process more than one unnamed inline data file, the results cannot be predicted if the wrong unnamed inline data file is used.

Open Considerations for Inline Data Files

The following considerations apply to opening inline data files:

- Record length specifies the length of the input records. (Record length is optional.) When the record length exceeds the length of the data, a message is sent to your program. The data is padded with blanks. When the record length is less than the data length, the records are truncated.
- When a file is specified in a program, the system searches for the file as a named inline data file before it searches for the file in a library. Therefore, if a named inline data file has the same name as a file that is not an inline data file, the inline data file is always used, even if the file name is qualified by a library name.
- Named inline data files *can* be shared between programs in the same job by specifying SHARE(*YES) on a create file or override file command.

  For example, if an override file command specifying a file named INPUT and SHARE(*YES) is in a batch job with an inline data file named INPUT, any programs running in the job that specify the file name INPUT will share the same named inline data file.

  Unnamed inline data files *cannot* be shared between programs in the same job.
- When you use inline data files, you should make sure the correct file type is specified on the //DATA command. For example, if the file is to be used as a source file, the file type on the //DATA command must be source.
- Inline data files must be opened for input only.

Spooling Subsystem

The spooling subsystem, QSPL, is used for processing the spooling readers and writers. The subsystem needs to be active when readers or writers are active. The spooling subsystem and the individual readers and writers can be controlled from jobs that run in other subsystems.

The start reader and start writer commands submit jobs to the job queue of the spooling subsystem.

Requests for reader or writer jobs are placed on the QSPL job queue, and the next entry on the QSPL job queue is selected to run if:

- The number of active jobs is less than the QSPL subsystem attribute of MAXJOBS.
- The number of active jobs from the QSPL job queue is less than the MAXACT attribute for the job queue.

Work management associated with the QSPL subsystem is similar to that for other subsystems as described in the *Work Management* book. To control the storage used on your system see “Spooling Library” on page 151.

Spooling Library

The spooling library (QSPL) contains database files that are used to store data for inline data files and spooled files. Each file in library QSPL can have several members. Each member contains all the data for an inline data file or spooled file.

When the spooled file is printed or deleted, its associated database member in the spooling library is cleared of records, but not removed, so that it can be used for another inline data file or spooled file. If no database members are available in library QSPL, then a member is automatically created.

Printing a spooled file or clearing an output queue does not reduce the number of associated database members. If an excessive number of associated database members were created on your system (for example, if a program went into a loop and created thousands of spooled files), the spool database members use storage on the system even if you clear the output queue.

Because the system keeps the date and time whenever a database member becomes available (for example, clearing of records after the spooled file has been printed or deleted), you can remove these spooled database members in the following ways:

- QRCLSPLSTG system value

  When this system value is set, the system removes spool database members that have been available for more than the number of days specified by the system value. The default value is 8 days. Values that can be set for this system value are:

  – 1-366: Valid range of day values that can be set. When an available member is older than the set number of days, it is removed by the system.

  – *NOMAX: Available spool database members are never automatically removed. The user must use the Reclaim Spool Storage (RCLSPLSTG) command to remove these members.

  – *NONE: The database member is removed as soon as the spooled file is printed or deleted.

    Note: If *NONE is specified, you will never have available database members in QSPL. If there are no available members when subsequent inline data files or spooled files are created, the system creates members and allocates storage to be used. This slows down the jobs that are creating inline data files or spooled files. It is highly recommended that the system value never be set to *NONE.

- RCLSPLSTG command

  Removes available database members that have been cleared of records for more than the number of days specified on the command. The command will run until it completes in the user's process.

The procedures previously described are the only allowable ways to remove spooled files from the QSPL library. Any other way can cause severe problems. It is best to keep the QSPL library small by periodically deleting old spooled files with the DLTSPLF or CLROUTQ commands. This procedure allows database members to be used again, rather than having to increase the size of the spooling library to accommodate new database members.

Displaying the data in the QSPL library may also prevent the data from being cleared, wasting storage space. Any command or program used to look at a

database file in the QSPL library must allocate the database file and member; if a writer tries to remove an allocated member after printing is completed, it will not be able to clear the member. Because the member is not cleared, it cannot be used for another inline data file or spooled file and it will not be removed by setting the QRCLSPLSTG system value or running the RCLSPLSTG command.

Saving a database file in the QSPL library can cause more problems than displaying the data in one member of the file because all members will be allocated a much longer time when a database file is saved. Because restoring these files destroys present and future spooled file data, there is no reason to save one of these files.

The QSPL library type and authority should not be changed. The authority to the files within QSPL should also not be changed. The QSPL library and the files in it are created in a particular way so that system spooling functions can access them. Changing the library or files could cause some system spooling functions to work incorrectly.

Appendix A. Feedback Area Layouts

Tables in this section describe the open and I/O feedback areas associated with any opened file. The following information is presented for each item in these feedback areas:

- Offset, which is the number of bytes from the start of the feedback area to the location of each item.
- Data Type.
- Length, which is given in number of bytes.
- Contents, which is the description of the item and the valid values for it.
- File type, which is an indication of what file types each item is valid for.

The support provided by the high-level language you are using determines how to access this information and how the data types are represented. See your high-level language information for more information.

Open Feedback Area

The open feedback area is the part of the open data path (ODP) that contains general information about the file after it has been opened. It also contains file-specific information, depending on the file type, plus information about each device or communications session defined for the file. This information is set during open processing and may be updated as other operations are performed.

Table 16. Open Feedback Area

Offset	Data Type	Length	Contents	File Type
0	Character	2	Open data path (ODP) type: **DS** Display, tape, ICF, save, printer file not being spooled, or diskette file not being spooled. **DB** Database member. **SP** Printer or diskette file being spooled or inline data file.	All
2	Character	10	Name of the file being opened. If the ODP type is DS, this is the name of the device file or save file. If the ODP type is SP, this is the name of the device file or the inline data file. If the ODP type is DB, this is the name of the database file that the member belongs to.	All
12	Character	10	Name of the library containing the file. For an inline data file, the value is *N.	All
22	Character	10	Name of the spooled file. The name of a database file containing the spooled input or output records.	Printer or diskette being spooled or inline data
32	Character	10	Name of the library in which the spooled file is located.	Printer or diskette being spooled or inline data
42	Binary	2	Spooled file number.	Printer or diskette being spooled
44	Binary	2	Maximum record length.	All
46	Binary	2	Maximum key length.	Database
48	Character	10	Member name: • If ODP type DB, the member name in the file named at offset 2. If file is overridden to MBR(*ALL), the member name that supplied the last record. • If ODP type SP, the member name in the file named at offset 22.	Database, printer, diskette, and inline data
58	Binary	4	Reserved.	
62	Binary	4	Reserved.	

Table 16. Open Feedback Area (continued)

Offset	Data Type	Length	Contents		File Type
66	Binary	2	File type:		All
			1	Display	
			2	Printer	
			4	Diskette	
			5	Tape	
			9	Save	
			10	DDM	
			11	ICF	
			20	Inline data	
			21	Database	
68	Character	3	Reserved.		
71	Binary	2	Number of lines on a display screen or number of lines on a printed page.		Display, printer
			Length of the null field byte map.		Database
73	Binary	2	Number of positions on a display screen or number of characters on a printed line.		Display, printer
			Length of the null key field byte map.		Database
75	Binary	4	Number of records in the member at open time. For a join logical file, the number of records in the primary. Supplied only if the file is being opened for input.		Database, inline data
79	Character	2	Access type:		Database
			AR	Arrival sequence.	
			KC	Keyed with duplicate keys allowed. Duplicate keys are accessed in first-changed-first-out (FCFO) order.	
			KF	Keyed with duplicate keys allowed. Duplicate keys are accessed in first-in-first-out (FIFO) order.	
			KL	Keyed with duplicate keys allowed. Duplicate keys are accessed in last-in-first-out (LIFO) order.	
			KN	Keyed with duplicate keys allowed. The order in which duplicate keys are accessed can be one of the following: • First-in-first-out (FIFO) • Last-in-first-out (LIFO) • First-changed-first-out (FCFO)	
			KU	Keyed, unique.	
81	Character	1	Duplicate key indication. Set only if the access path is KC, KF, KL, KN, or KU:		Database
			D	Duplicate keys allowed if the access path is KF or KL.	
			U	Duplicate keys are not allowed; all keys are unique and the access path is KU.	
82	Character	1	Source file indication.		Database, tape, diskette, and inline data
			Y	File is a source file.	
			N	File is not a source file.	
83	Character	10	Reserved.		
93	Character	10	Reserved.		
103	Binary	2	Offset to volume label fields of open feedback area.		Diskette, tape
105	Binary	2	Maximum number of records that can be read or written in a block when using blocked record I/O.		All
107	Binary	2	Overflow line number.		Printer
109	Binary	2	Blocked record I/O record increment. Number of bytes that must be added to the start of each record in a block to address the next record in the block.		All
111	Binary	4	Reserved.		

Table 16. Open Feedback Area (continued)

Offset	Data Type	Length	Contents	File Type
115	Character	1	Miscellaneous flags.	
			Bit 1: Reserved.	
			Bit 2: File shareable	All
			0 File was not opened shareable.	
			1 File was opened shareable (SHARE(*YES)).	
			Bit 3: Commitment control	Database
			0 File is not under commitment control.	
			1 File is under commitment control.	
			Bit 4: Commitment lock level	Database
			0 Only changed records are locked (LCKLVL (*CHG)). If this bit is zero and bit 8 of the character at offset 132 is one, then all records accessed are locked, but the locks are released when the current position in the file changes (LCKLVL (*CS)).	
			1 All records accessed are locked (LCKLVL (*ALL)).	
			Bit 5: Member type	Database
			0 Member is a physical file member.	
			1 Member is a logical file member.	
			Bit 6: Field-level descriptions	All, except database
			0 File does not contain field-level descriptions.	
			1 File contains field-level descriptions.	
			Bit 7: DBCS or graphic-capable file	Database, display, printer, tape, diskette, and ICF
			0 File does not contain DBCS or graphic-capable fields.	
			1 File does contain DBCS or graphic-capable fields.	
			Bit 8: End-of-file delay	Database
			0 End-of-file delay processing is not being done.	
			1 End-of-file delay processing is being done.	
116	Character	10	Name of the requester device. For display files, this is the name of the display device description that is the requester device. For ICF files, this is the program device name associated with the remote location of *REQUESTER. This field is supplied only when either a device or remote location name of *REQUESTER is being attached to the file by an open or acquire operation. Otherwise, this field contains *N.	Display, ICF
126	Binary	2	File open count. If the file has not been opened shareable, this field contains a 1. If the file has been opened shareable, this field contains the number of programs currently attached to this file.	All
128	Binary	2	Reserved.	
130	Binary	2	Number of based-on physical members opened. For logical members, this is the number of physical members over which the logical member was opened. For physical members, this field is always set to 1.	Database
132	Character	1	Miscellaneous flags.	

Table 16. Open Feedback Area (continued)

Offset	Data Type	Length	Contents		File Type
			Bit 1: Multiple member processing		Database
			0	Only the member specified will be processed.	
			1	All members will be processed.	
			Bit 2: Join logical file		Database
			0	File is not a join logical file.	
			1	File is a join logical file.	
			Bit 3: Local or remote data (DDM files)		Database
			0	Data is stored on local system.	
			1	Data is stored on remote system.	
			Bit 4: Remote System/38 or AS/400 data (DDM files). Applicable only if the value of Bit 3 is 1.		Database
			0	Data is on a remote System/38 or AS/400 system.	
			1	Data is not on a remote System/38 or AS/400 system.	
			Bit 5: Separate indicator area		Printer, display, and ICF
			0	Indicators are in the I/O buffer of the program.	
			1	Indicators are not in the I/O buffer of the program. The DDS keyword, INDARA, was used when the file was created.	
			Bit 6: User buffers		All
			0	System creates I/O buffers for the program.	
			1	User program supplies I/O buffers.	
			Bit 7: Reserved.		
			Bit 8: Additional commitment lock level indicator. This is only valid if bit 3 of the character at offset 115 is one.		Database
			If bit 4 of the character at offset 115 is zero:		
			0	Only changed records are locked (LCKLVL(*CHG)).	
			1	All records accessed are locked, but the locks are released when the current position in the file changes (LCKLVL(*CS)).	
			If bit 4 of the character at offset 115 is one:		
			0	All records accessed are locked (LCKLVL(*ALL)).	
			1	Reserved.	
133	Character	2	Open identifier. This value is unique for a full open operation (SHARE(*NO)) or the first open of a file that is opened with SHARE(*YES). This is used for display and ICF files, but is set up for all file types. It allows you to match this file to an entry on the associated data queue.		All
135	Binary	2	The field value is the maximum record format length, including both data and file-specific information such as: first-character forms control, option indicators, response indicators, source sequence numbers, and program-to-system data. If the value is zero, then use the field at offset 44.		Printer, diskette, tape, and ICF
137	Binary	2	Coded character set identifier (CCSID) of the character data in the buffer.		Database

Table 16. Open Feedback Area (continued)

Offset	Data Type	Length	Contents	File Type
139	Character	1	Miscellaneous flags.	Database
			Bit 1: Null-capable field file.	
			0 File does not contain null-capable fields.	
			1 File contains null-capable fields.	
			Bit 2: Variable length fields file.	Database
			0 File does not contain any variable length fields.	
			1 File contains variable length fields.	
			Bit 3: Variable length record processing	Database
			0 Variable length record processing will not be done.	
			1 Variable length record processing will be done.	
			Bit 4: CCSID character substitution	Database, Display
			0 No substitution characters will be used during CCSID data conversion.	
			1 Substitution characters may be used during CCSID data conversion.	
			Bit 5: Job Level Open Indicator	All
			0 This ODP is not scoped to the job level.	
			1 This ODP is scoped to the job level.	
			Bits 6-8: Reserved.	
140	Character	6	Reserved.	
146	Binary	2	Number of devices defined for this ODP. For displays, this is determined by the number of devices defined on the DEV parameter of the Create Display File (CRTDSPF) command. For ICF, this is determined by the number of program devices defined or acquired with the Add ICF Device Entry (ADDICFDEVE) or the Override ICF Device Entry (OVRICFDEVE) command. For all other files, it has the value of 1.	All
148	Character		Device name definition list. See “Device Definition List” for a description of this array.	All

Device Definition List

The device definition list part of the open feedback area is an array structure. Each entry in the array contains information about each device or communications session attached to the file. The number of entries in this array is determined by the number at offset 146 of the open feedback area. The device definition list begins at offset 148 of the open feedback area. The offsets shown for it are from the start of the device definition list rather than the start of the open feedback area.

Table 17. Device Definition List

Offset	Data Type	Length	Contents	File Type
0	Character	10	Program device name. For database files, the value is DATABASE. For printer or diskette files being spooled, the value is *N. For save files, the value is *NONE. For ICF files, the value is the name of the program device from the ADDICFDEVE or OVRICFDEVE command. For all other files, the value is the name of the device description.	All, except inline data
10	Character	50	Reserved.	

Table 17. Device Definition List (continued)

Offset	Data Type	Length	Contents	File Type
60	Character	10	Device description name. For printer or diskette files being spooled, the value is *N. For save files, the value is *NONE. For all other files, the value is the name of the device description.	All, except database and inline data
70	Character	1	Device class. **hex 01** Display **hex 02** Printer **hex 04** Diskette **hex 05** Tape **hex 09** Save **hex 0B** ICF	All, except database and inline data
71	Character	1	Device type. **hex 02** 5256 Printer **hex 07** 5251 Display Station **hex 08** Spooled **hex 0A** BSCEL **hex 0B** 5291 Display Station **hex 0C** 5224/5225 printers **hex 0D** 5292 Display Station **hex 0E** APPC **hex 0F** 5219 Printer **hex 10** 5583 Printer (DBCS) **hex 11** 5553 Printer **hex 12** 5555-B01 Display Station **hex 13** 3270 Display Station **hex 14** 3270 Printer **hex 15** Graphic-capable device **hex 16** Financial Display Station **hex 17** 3180 Display Station **hex 18** Save file **hex 19** 3277 DHCF Device **hex 1A** 9347 Tape Unit **hex 1B** 9348 Tape Unit **hex 1C** 9331-1 Diskette Unit **hex 1D** 9331-2 Diskette Unit **hex 1E** Intrasystem communications support **hex 1F** Asynchronous communications support	

Table 17. Device Definition List (continued)

Offset	Data Type	Length	Contents		File Type
			hex 20	SNUF	
			hex 21	4234 (SCS) Printer	
			hex 22	3812 (SCS) Printer	
			hex 23	4214 Printer	
			hex 24	4224 (IPDS) Printer	
			hex 25	4245 Printer	
			hex 26	3179-2 Display Station	
			hex 27	3196-A Display Station	
			hex 28	3196-B Display Station	
			hex 29	5262 Printer	
			hex 2A	6346 Tape Unit	
			hex 2B	2440 Tape Unit	
			hex 2C	9346 Tape Unit	
			hex 2D	6331 Diskette Unit	
			hex 2E	6332 Diskette Unit	
			hex 30	3812 (IPDS) Printer	
			hex 31	4234 (IPDS) Printer	
			hex 32	IPDS printer, model unknown	
			hex 33	3197-C1 Display Station	
			hex 34	3197-C2 Display Station	
			hex 35	3197-D1 Display Station	
			hex 36	3197-D2 Display Station	
			hex 37	3197-W1 Display Station	
			hex 38	3197-W2 Display Station	
			hex 39	5555-E01 Display Station	
			hex 3A	3430 Tape Unit	
			hex 3B	3422 Tape Unit	
			hex 3C	3480 Tape Unit	
			hex 3D	3490 Tape Unit	
			hex 3E	3476-EA Display Station	
			hex 3F	3477-FG Display Station	

Table 17. Device Definition List (continued)

Offset	Data Type	Length	Contents	File Type
			hex 40 3278 DHCF device	
			hex 41 3279 DHCF device	
			hex 42 ICF finance device	
			hex 43 Retail communications device	
			hex 44 3477-FA Display Station	
			hex 45 3477-FC Display Station	
			hex 46 3477-FD Display Station	
			hex 47 3477-FW Display Station	
			hex 48 3477-FE Display Station	
			hex 49 6367 Tape Unit	
			hex 4A 6347 Tape Unit	
			hex 4D Network Virtual Terminal Display Station	
			hex 4E 6341 Tape Unit	
			hex 4F 6342 Tape Unit	
			hex 50 6133 Diskette Unit	
			hex 51 5555-C01 Display Station	
			hex 52 5555-F01 Display Station	
			hex 53 6366 Tape Unit	
			hex 54 7208 Tape Unit	
			hex 55 6252 (SCS) Printer	
			hex 56 3476-EC Display Station	
			hex 57 4230 (IPDS) Printer	
			hex 58 5555-G01 Display Station	
			hex 59 5555-G02 Display Station	
			hex 5A 6343 Tape Unit	
			hex 5B 6348 Tape Unit	
			hex 5C 6368 Tape Unit	
			hex 5D 3486-BA Display Station	
			hex 5F 3487-HA Display Station	

Table 17. Device Definition List (continued)

Offset	Data Type	Length	Contents	File Type
			hex 60 3487-HG Display Station	
			hex 61 3487-HW Display Station	
			hex 62 3487-HC Display Station	
			hex 63 3935 (IPDS) Printer	
			hex 64 6344 Tape Unit	
			hex 65 6349 Tape Unit	
			hex 66 6369 Tape Unit	
			hex 67 6380 Tape Unit	
			hex 68 6378 Tape Unit	
			hex 69 6390 Tape Unit	
			hex 70 6379 Tape Unit	
			hex 71 9331-11 Diskette Unit	
			hex 72 9331-12 Diskette Unit	
			hex 73 3570 Tape Unit	
			hex 74 3590 Tape Unit	
			hex 75 6335 Tape Unit	
72	Binary	2	Number of lines on the display screen.	Display
74	Binary	2	Number of positions in each line of the display screen.	Display
76	Character	2	Bit flags.	Display
			Bit 1: Blinking capability.	
			0 Display is not capable of blinking.	
			1 Display is capable of blinking.	
			Bit 2: Device location.	Display
			0 Local device.	
			1 Remote device.	
			Bit 3: Acquire status. This bit is set even if the device is implicitly acquired at open time.	Display, ICF
			0 Device is not acquired.	
			1 Device is acquired.	
			Bit 4: Invite status.	Display, ICF
			0 Device is not invited.	
			1 Device is invited.	
			Bit 5: Data available status (only if device is invited).	Display, ICF
			0 Data is not available.	
			1 Data is available.	

Table 17. Device Definition List (continued)

Offset	Data Type	Length	Contents	File Type
			Bit 6: Transaction status. **0** Transaction is not started. An evoke request has not been sent, a detach request has been sent or received, or the transaction has completed. **1** Transaction is started. The transaction is active. An evoke request has been sent or received and the transaction has not ended.	ICF
			Bit 7: Requester device. **0** Not a requester device. **1** A requester device.	Display, ICF
			Bit 8: DBCS device. **0** Device is not capable of processing double-byte data. **1** Device is capable of processing double-byte data.	Display
			Bits 9-10: Reserved.	
			Bit 11: DBCS keyboard. **0** Keyboard is not capable of entering double-byte data. **1** Keyboard is capable of entering double-byte data.	Display
			Bits 12-16: Reserved.	
78	Character	1	Synchronization level. **hex 00** The transaction was built with SYNLVL(*NONE). Confirm processing is not allowed. **hex 01** The transaction was built with SYNLVL(*CONFIRM). Confirm processing is allowed. **hex 02** The transaction was built with SYNLVL(*COMMIT).	ICF
79	Character	1	Conversation type. **hex D0** Basic conversation (CNVTYPE(*USER)). **hex D1** Mapped conversation (CNVTYPE(*SYS)).	ICF
80	Character	50	Reserved.	

Volume Label Fields

Table 18. Volume Label Fields

Offset	Data Type	Length	Contents	File Type
0	Character	128	Volume label of current volume.	Diskette, tape
128	Character	128	Header label 1 of the opened file.	Diskette, tape
256	Character	128	Header label 2 of the opened file.	Tape

I/O Feedback Area

AS/400 uses OS/400 messages and I/O feedback information to communicate the results of I/O operations to the program. The system updates the I/O feedback area for every successful I/O operation unless your program uses blocked record I/O. In that case, the system updates the feedback area only when it reads or writes a block of records. Some of the information reflects the last record in the block. Other information, such as the count of I/O operations, reflects the number of operations on blocks of records and not the number of records. See your high-level language information to determine if your program uses blocked record I/O.

The I/O feedback area consists of two parts: a common area and a file-dependent area. The file-dependent area varies by the file type.

Common I/O Feedback Area

Table 19. Common I/O Feedback Area

Offset	Data Type	Length	Contents
0	Binary	2	Offset to file-dependent feedback area.
2	Binary	4	Write operation count. Updated only when a write operation completes successfully. For blocked record I/O operations, this count is the number of blocks, not the number of records.
6	Binary	4	Read operation count. Updated only when a read operation completes successfully. For blocked record I/O operations, this count is the number of blocks, not the number of records.
10	Binary	4	Write-read operation count. Updated only when a write-read operation completes successfully.
14	Binary	4	Other operation count. Number of successful operations other than write, read, or write-read. Updated only when the operation completes successfully. This count includes update, delete, force-end-of-data, force-end-of-volume, change-end-of-data, release record lock, and acquire/release device operations.
18	Character	1	Reserved.

Table 19. Common I/O Feedback Area (continued)

Offset	Data Type	Length	Contents
19	Character	1	Current operation. **hex 01** Read or read block or read from invited devices **hex 02** Read direct **hex 03** Read by key **hex 05** Write or write block **hex 06** Write-read **hex 07** Update **hex 08** Delete **hex 09** Force-end-of-data **hex 0A** Force-end-of-volume **hex 0D** Release record lock **hex 0E** Change end-of-data **hex 0F** Put deleted record **hex 11** Release device **hex 12** Acquire device
20	Character	10	Name of the record format just processed, which is either: • Specified on the I/O request, or • Determined by default or format selection processing For display files, the default name is either the name of the only record format in the file or the previous record format name for the record written to the display that contains input-capable fields. Because a display file may have multiple formats on the display at the same time, this format may not represent the format where the last cursor position was typed. For ICF files, the format name is determined by the system, based on the format selection option used. Refer to the *ICF Programming* book for more information.
30	Character	2	Device class: Byte 1: **hex 00** Database **hex 01** Display **hex 02** Printer **hex 04** Diskette **hex 05** Tape **hex 09** Save **hex 0B** ICF

Table 19. Common I/O Feedback Area (continued)

Offset	Data Type	Length	Contents
			Byte 2 (if byte 1 contains hex 00): **hex 00** Nonkeyed file **hex 01** Keyed file Byte 2 (if byte 1 does not contain hex 00): **hex 02** 5256 Printer **hex 07** 5251 Display Station **hex 08** Spooled **hex 0A** BSCEL **hex 0B** 5291 Display Station **hex 0C** 5224/5225 printers **hex 0D** 5292 Display Station **hex 0E** APPC **hex 0F** 5219 Printer **hex 10** 5583 Printer (DBCS) **hex 11** 5553 Printer **hex 12** 5555-B01 Display Station **hex 13** 3270 Display Station **hex 14** 3270 Printer **hex 15** Graphic-capable device **hex 16** Financial Display Station **hex 17** 3180 Display Station **hex 18** Save file **hex 19** 3277 DHCF device **hex 1A** 9347 Tape Unit **hex 1B** 9348 Tape Unit **hex 1C** 9331-1 Diskette Unit **hex 1D** 9331-2 Diskette Unit **hex 1E** Intrasystem communications support **hex 1F** Asynchronous communications support

Table 19. Common I/O Feedback Area (continued)

Offset	Data Type	Length	Contents
			hex 20 SNUF
			hex 21 4234 (SCS) Printer
			hex 22 3812 (SCS) Printer
			hex 23 4214 Printer
			hex 24 4224 (IPDS) Printer
			hex 25 4245 Printer
			hex 26 3179-2 Display Station
			hex 27 3196-A Display Station
			hex 28 3196-B Display Station
			hex 29 5262 Printer
			hex 2A 6346 Tape Unit
			hex 2B 2440 Tape Unit
			hex 2C 9346 Tape Unit
			hex 2D 6331 Diskette Unit
			hex 2E 6332 Diskette Unit
			hex 30 3812 (IPDS) Printer
			hex 31 4234 (IPDS) Printer
			hex 32 IPDS printer, model unknown
			hex 33 3197-C1 Display Station
			hex 34 3197-C2 Display Station
			hex 35 3197-D1 Display Station
			hex 36 3197-D2 Display Station
			hex 37 3197-W1 Display Station
			hex 38 3197-W2 Display Station
			hex 39 5555-E01 Display Station
			hex 3A 3430 Tape Unit
			hex 3B 3422 Tape Unit
			hex 3C 3480 Tape Unit
			hex 3D 3490 Tape Unit
			hex 3E 3476-EA Display Station
			hex 3F 3477-FG Display Station

Table 19. Common I/O Feedback Area (continued)

Offset	Data Type	Length	Contents
			hex 40 3278 DHCF device
			hex 41 3279 DHCF device
			hex 42 ICF finance device
			hex 43 Retail communications device
			hex 44 3477-FA Display Station
			hex 45 3477-FC Display Station
			hex 46 3477-FD Display Station
			hex 47 3477-FW Display Station
			hex 48 3477-FE Display Station
			hex 49 6367 Tape Unit
			hex 4A 6347 Tape Unit
			hex 4D Network Virtual Terminal Display Station
			hex 4E 6341 Tape Unit
			hex 4F 6342 Tape Unit
			hex 50 6133 Diskette Unit
			hex 51 5555-C01 Display Station
			hex 52 5555-F01 Display Station
			hex 53 6366 Tape Unit
			hex 54 7208 Tape Unit
			hex 55 6252 (SCS) Printer
			hex 56 3476-EC Display Station
			hex 57 4230 (IPDS) Printer
			hex 58 5555-G01 Display Station
			hex 59 5555-G02 Display Station
			hex 5A 6343 Tape Unit
			hex 5B 6348 Tape Unit
			hex 5C 6368 Tape Unit
			hex 5D 3486-BA Display Station
			hex 5F 3487-HA Display Station

Table 19. Common I/O Feedback Area (continued)

Offset	Data Type	Length	Contents
			hex 60 3487-HG Display Station **hex 61** 3487-HW Display Station **hex 62** 3487-HC Display Station **hex 63** 3935 (IPDS) Printer **hex 64** 6344 Tape Unit **hex 65** 6349 Tape Unit **hex 66** 6369 Tape Unit **hex 67** 6380 Tape Unit **hex 68** 6378 Tape Unit **hex 69** 6390 Tape Unit **hex 70** 6379 Tape Unit **hex 71** 9331-11 Diskette Unit **hex 72** 9331-12 Diskette Unit **hex 73** 3570 Tape Unit **hex 74** 3590 Tape Unit **hex 75** 6335 Tape Unit
32	Character	10	Device name. The name of the device for which the operation just completed. Supplied only for display, printer, tape, diskette, and ICF files. For printer or diskette files being spooled, the value is *N. For ICF files, the value is the program device name. For other files, the value is the device description name.
42	Binary	4	Length of the record processed by the last I/O operation (supplied only for an ICF, display, tape, or database file). On ICF write operations, this is the record length of the data. On ICF read operations, it is the record length of the record associated with the last read operation.
46	Character	80	Reserved.
126	Binary	2	Number of records retrieved on a read request for blocked records or sent on a write or force-end-of-data or force-end-of-volume request for blocked records. Supplied only for database, diskette, and tape files.
128	Binary	2	For output, the field value is the record format length, including first-character forms control, option indicators, source sequence numbers, and program-to-system data. If the value is zero, use the field at offset 42. For input, the field value is the record format length, including response indicators and source sequence numbers. If the value is zero, use the field at offset 42.
130	Character	2	Reserved.
132	Binary	4	Current block count. The number of blocks of the tape data file already written or read. For tape files only.
136	Character	8	Reserved.

I/O Feedback Area for ICF and Display Files

Table 20. I/O Feedback Area for ICF and Display Files

Offset	Data Type	Length	Contents	File Type
0	**Character**	**2**	**Flag bits.** **Bit 1:** Cancel-read indicator. **0** The cancel-read operation did not cancel the read request. **1** The cancel-read operation canceled the read request. **Bit 2:** Data-returned indicator. **0** The cancel-read operation did not change the contents of the input buffer. **1** The cancel-read operation placed the data from the read-with-no-wait operation into the input buffer. **Bit 3:** Command key indicator. **0** Conditions for setting this indicator did not occur. **1** The Print, Help, Home, Roll Up, Roll Down, or Clear key was pressed. The key is enabled with a DDS keyword, but without a response indicator specified. **Bits 4-16:** Reserved.	**Display**

Table 20. I/O Feedback Area for ICF and Display Files (continued)

Offset	Data Type	Length	Contents	File Type
2	**Character**	**1**	**Attention indicator byte (AID). This field identifies which function key was pressed.** For ICF files, this field will always contain the value hex F1 to imitate the Enter key being pressed on a display device. For display files, this field will contain the 1-byte hexadecimal value returned from the device. **Hex Codes** **Function Keys** **hex 31** 1 **hex 32** 2 **hex 33** 3 **hex 34** 4 **hex 35** 5 **hex 36** 6 **hex 37** 7 **hex 38** 8 **hex 39** 9 **hex 3A** 10 **hex 3B** 11 **hex 3C** 12 **hex B1** 13 **hex B2** 14 **hex B3** 15 **hex B4** 16 **hex B5** 17 **hex B6** 18 **hex B7** 19 **hex B8** 20 **hex B9** 21 **hex BA** 22 **hex BB** 23 **hex BC** 24 **hex BD** Clear **hex F1** Enter/Rec Adv **hex F3** Help (not in operator-error mode) **hex F4** Roll Down **hex F5** Roll Up **hex F6** Print **hex F8** Record Backspace **hex 3F** Auto Enter (for Selector Light Pen)	**Display, ICF**

Table 20. I/O Feedback Area for ICF and Display Files (continued)

Offset	Data Type	Length	Contents	File Type
3	**Character**	**2**	**Cursor location (line and position). Updated on input operations that are not subfile operations that return data to the program. For example, hex 0102 means line 1, position 2. Line 10, position 33 would be hex 0A21.**	**Display**
5	**Binary**	**4**	**Actual data length. For an ICF file, see the** *ICF Programming* book for additional information. For a display file, this is the length of the record format processed by the I/O operation.	**Display, ICF**
9	**Binary**	**2**	**Relative record number of a subfile record. Updated for a subfile record operation. For input operations, updated only if data is returned to the program. If multiple subfiles are on the display, this offset will contain the relative record number for the last subfile updated.**	**Display**
11	**Binary**	**2**	**Indicates the lowest subfile relative record number currently displayed in the uppermost subfile display area if the last write operation was done to the subfile control record with SFLDSP specified. Updated for roll up and roll down operations. Reset to 0 on a write operation to another record. Not set for message subfiles.**	**Display**
13	**Binary**	**2**	**Total number of records in a subfile. Updated on a put-relative operation to any subfile record. The number is set to zero on a write or write-read operation to any subfile control record with the SFLINZ keyword optioned on. If records are put to multiple subfiles on the display, this offset will contain the total number of records for all subfiles assuming that no write or write-read operations were performed to any subfile control record with the SFLINZ keyword optioned on.**	**Display**
15	**Character**	**2**	**Cursor location (line and position) within active window. Updated on input operations that are not subfile operations that return data to the program. For example, hex 0203 means line 2, position 3 relative to the upper-left corner of the active window.**	**Display**
17	**Character**	**17**	**Reserved.**	

Table 20. I/O Feedback Area for ICF and Display Files (continued)

Offset	Data Type	Length	Contents		File Type
34	**Character**	**2**	**Major return code.**		**Display, ICF**
			00	Operation completed successfully.	
			02	Input operation completed successfully, but job is being canceled (controlled).	
			03	Input operation completed successfully, but no data received.	
			04	Output exception.	
			08	Device already acquired.	
			11	Read from invited devices was not successful.	
			34	Input exception.	
			80	Permanent system or file error.	
			81	Permanent session or device error.	
			82	Acquire or open operation failed.	
			83	Recoverable session or device error.	
36	**Character**	**2**	**Minor return code. For the values for a display file, see the** *Application Display Programming* book. For the values for an ICF file, see the *ICF Programming* book and the appropriate communications-type programmer's guide.		**Display, ICF**
38	**Character**	**8**	**Systems Network Architecture (SNA) sense return code. For some return codes, this field may contain more detailed information about the reason for the error. For a description of the SNA sense codes, see the appropriate SNA book.**		**ICF**
46	**Character**	**1**	**Safe indicator:**		**ICF**
			0	An end-of-text (ETX) control character has not been received.	
			1	An ETX control character has been received.	
47	**Character**	**1**	**Reserved.**		
48	**Character**	**1**	**Request Write (RQSWRT) command from remote system/application.**		**ICF**
			0	RQSWRT not received	
			1	RQSWRT received	
49	**Character**	**10**	**Record format name received from the remote system.**		**ICF**
59	**Character**	**4**	**Reserved.**		
63	**Character**	**8**	**Mode name.**		**ICF**
71	**Character**	**9**	**Reserved.**		

I/O Feedback Area for Printer Files

Table 21. I/O Feedback Area for Printer Files

Offset	Data Type	Length	Contents
0	Binary	2	Current line number in a page.
2	Binary	4	Current page count.
6	Character	1	Bit 1: Spooled file has been deleted: **1** The spooled file has been deleted. **0** The spooled file has not been deleted. Bits 2 - 8: Reserved.
7	Character	27	Reserved.
34	Character	2	Major return code. **00** Operation completed successfully. **80** Permanent system or file error. **81** Permanent device error. **82** Open operation failed. **83** Recoverable device error occurred.
36	Character	2	Minor return code. For the values for a printer file, refer to the *Printer Device Programming*.

I/O Feedback Area for Database Files

Table 22. I/O Feedback Area for Database Files

Offset	Data Type	Length	Contents
0	Binary	4	Size of the database feedback area, including the key and the null key field byte map.
4	Character	4	**Bits 1-32:** Each bit represents a join logical file in JFILE keyword. **0** JDFTVAL not supplied for file **1** JDFTVAL supplied for file
8	Binary	2	Offset from the beginning of the I/O feedback area for database files to the null key field byte map which follows the key value (which begins at offset 34 in this area).
10	Binary	2	Number of locked records.
12	Binary	2	Maximum number of fields.
14	Binary	4	Offset to the field-mapping error-bit map.
18	Character	1	Current file position indication. **Bit 1:** Current file position is valid for get-next-key equal operation. **0** File position is not valid. **1** File position is valid. **Bits 2-8:** Reserved.

Table 22. I/O Feedback Area for Database Files (continued)

Offset	Data Type	Length	Contents
19	Character	1	Current record deleted indication: **Bits 1-2:** Reserved. **Bit 3:** Next message indicator. **0** Next message not end of file. **1** Next message may be end of file. **Bit 4:** Deleted record indicator. **0** Current file position is at an active record. **1** Current file position is at a deleted record. **Bit 5:** Write operation key feedback indicator. **0** Key feedback is not provided by last write operation. **1** Key feedback is provided by last write operation. **Bit 6:** File position changed indicator. Set only for read and positioning I/O operations. Not set for write, update, and delete I/O operations. **0** File position did not change. **1** File position did change. **Bit 7:** Pending exception indicator. Valid for files open for input only and SEQONLY(*YES N) where N is greater than 1. **0** Pending retrieval error does not exist. **1** Pending retrieval error does exist. **Bit 8:** Duplicate key indicator. **0** The key of the last read or write operation was not a duplicate key. **1** The key of the last read or write operation was a duplicate key.
20	Binary	2	Number of key fields. Use this offset for binary operations. Use the next offset (offset 21) for character operations. These offsets overlap and provide the same value (there can be no more than 32 key fields, and only the low-order byte of offset 20 is used).
21	Character	1	Number of key fields.
22	Character	4	Reserved.
26	Binary	2	Key length.
28	Binary	2	Data member number.
30	Binary	4	Relative record number in data member.
34	Character	*	Key value.

Table 22. I/O Feedback Area for Database Files (continued)

Offset	Data Type	Length	Contents
*	Character	*	Null key field byte map.

Get Attributes

Performing the get attributes operation allows you to obtain certain information about a specific display device or ICF session.

Table 23. Get Attributes

Offset	Data Type	Length	Contents	File Type
0	Character	10	Program device name.	Display, ICF
10	Character	10	Device description name. Name of the device description associated with this entry.	Display, ICF
20	Character	10	User ID.	Display, ICF
30	Character	1	Device class:	Display, ICF
			D Display	
			I ICF	
			U Unknown	
31	Character	6	Device type:	
			3179 3179 Display Station	
			317902 3179-2 Display Station	
			3180 3180 Display Station	
			3196A 3196-A1/A2 Display Station	
			3196B 3196-B1/B2 Display Station	
			3197C1 3197-C1 Display Station	
			3197C2 3197-C2 Display Station	
			3197D1 3197-D1 Display Station	
			3197D2 3197-D2 Display Station	
			3197W1 3197-W1 Display Station	
			3197W2 3197-W2 Display Station	
			3270 3270 Display Station	
			3476EA 3476-EA Display Station	
			3476EC 3476-EC Display Station	
			3477FA 3477-FA Display Station	
			3477FC 3477-FC Display Station	
			3477FD 3477-FD Display Station	
			3477FE 3477-FE Display Station	
			3477FG 3477-FG Display Station	
			3477FW 3477-FW Display Station	
			525111 5251 Display Station	
			5291 5291 Display Station	
			5292 5292 Display Station	
			529202 5292-2 Display Station	

Table 23. Get Attributes (continued)

Offset	Data Type	Length	Contents	File Type
			5555B1 5555-B01 Display Station	Display, ICF
			5555C1 5555-C01 Display Station	
			5555E1 5555-E01 Display Station	
			5555F1 5555-F01 Display Station	
			5555G1 5555-G01 Display Station	
			5555G2 5555-G02 Display Station	
			DHCF77 3277 DHCF device	
			DHCF78 3278 DHCF device	
			DHCF79 3279 DHCF device	
			3486BA 3486-BA Display Station	Display, ICF
			3487HA 3487-HA Display Station	
			3487HC 3487-HC Display Station	
			3487HG 3487-HG Display Station	
			3487HW 3487-HW Display Station	
			APPC Advance program-to-program communications device	
			ASYNC Asynchronous communications device	
			BSC Bisynchronous communications device	
			BSCEL BSCEL communications device	
			FINANC ICF Finance communications device	
			INTRA Intrasystem communications device	
			LU1 LU1 communications device	
			RETAIL RETAIL communications device	
			SNUF SNA upline facility communications device	
37	Character	1	Requester device. This flag indicates whether this entry is defining a *REQUESTER device.	Display, ICF
			N Not a *REQUESTER device (communications source device).	
			Y A *REQUESTER device (communications target device).	
38	Character	1	Acquire status. Set even if device is implicitly acquired at open time.	Display, ICF
			N Device is not acquired.	
			Y Device is acquired.	
39	Character	1	Invite status.	Display, ICF
			Y Device is invited.	
			N Device is not invited.	

Table 23. Get Attributes (continued)

Offset	Data Type	Length	Contents	File Type
40	Character	1	Data available.	Display, ICF
			Y Invited data is available.	
			N Invited data is not available.	
41	Binary	2	Number of rows on display.	Display
43	Binary	2	Number of columns on display.	Display
45	Character	1	Display allow blink.	Display
			Y Display is capable of blinking.	
			N Display is not capable of blinking.	
46	Character	1	Online/offline status.	Display
			O Display is online.	
			F Display is offline.	
47	Character	1	Display location.	Display
			L Local display.	
			R Remote display.	
48	Character	1	Display type.	Display
			A Alphanumeric or Katakana.	
			I DBCS.	
			G Graphic DBCS.	
49	Character	1	Keyboard type of display.	Display
			A Alphanumeric or Katakana keyboard.	
			I DBCS keyboard.	
50	Character	1	Transaction status. All communication types.	ICF
			N Transaction is not started. An evoke request has not been sent, a detach request has been sent or received, or the transaction has completed.	
			Y Transaction is started. The transaction is active. An evoke request has been sent or received and the transaction has not ended.	
51	Character	1	Synchronization level. APPC and INTRA.	ICF
			0 Synchronization level 0 (SYNLVL(*NONE)).	
			1 Synchronization level 1 (SYNLVL(*CONFIRM)).	
			2 Synchronization level 2 (SYNLVL(*COMMIT)).	
52	Character	1	Conversation being used. APPC only.	ICF
			M Mapped conversation.	
			B Basic conversation.	
53	Character	8	Remote location name. All communication types.	ICF
61	Character	8	Local LU name. APPC only.	ICF
69	Character	8	Local network ID. APPC only.	ICF
77	Character	8	Remote LU name. APPC only.	ICF
85	Character	8	Remote network ID. APPC only.	ICF
93	Character	8	Mode. APPC only.	ICF

Table 23. Get Attributes (continued)

Offset	Data Type	Length	Contents	File Type
101	Character	1	Controller information.	Display
			N Display is not attached to a controller that supports an enhanced interface for nonprogrammable work stations.	
			1 Display is attached to a controller (type 1) that supports an enhanced interface for nonprogrammable work stations. See note.	
			2 Display is attached to a controller (type 2) that supports an enhanced interface for nonprogrammable work stations. See note.	
			3 Display is attached to a controller (type 3) that supports an enhanced interface for nonprogrammable work stations. See note.	
102	Character	1	Color capability of display.	Display
			Y Color display	
			N Monochrome display	
103	Character	1	Grid line support by display.	Display
			Y Display supports grid lines	
			N Display does not support grid lines	
104	Character	1		ICF
			hex 00 Reset state	
			hex 01 Send state	
			hex 02 Defer received state	
			hex 03 Defer deallocate state	
			hex 04 Receive state	
			hex 05 Confirm state	
			hex 06 Confirm send state	
			hex 07 Confirm deallocate state	
			hex 08 Commit state	
			hex 09 Commit send state	
			hex 0A Commit deallocate state	
			hex 0B Deallocate state	
			hex 0C Rollback required state	
105	Character	8	LU.6 Conversation Correlator	ICF
113	Character	31	Reserved	Display, ICF
Note: The following information is provided only for an Integrated Service Digital Network (ISDN) used in the ICF or remote display session. Also, not all of the information will be available if the area to receive it is too small.				
144	Binary	2	ISDN remote number length in bytes. Consists of the total of the lengths of the next three fields: ISDN remote numbering type, ISDN remote numbering plan, and the ISDN remote number. If the ISDN remote number has been padded on the right with blanks, the length of that padding is not included in this total. :p If ISDN is not used, this field contains 0.	Display, ICF

Table 23. Get Attributes (continued)

Offset	Data Type	Length	Contents	File Type
146	Character	2	ISDN remote numbering type (decimal). **00** Unknown. **01** International. **02** National. **03** Network-specific. **04** Subscriber. **06** Abbreviated.	Display, ICF
148	Character	2	ISDN remote numbering plan (decimal). **00** Unknown. **01** ISDN/Telephony. **03** Data. **04** Telex**. **08** National Standard. **09** Private.	Display, ICF
150	Character	40	The ISDN remote number in EBCDIC, padded on the right with blanks if necessary to fill the field.	Display, ICF
190	Character	4	Reserved.	Display, ICF
194	Binary	2	ISDN remote subaddress length in bytes. Consists of the total of the lengths of the next two fields: ISDN remote subaddress type and the ISDN remote subaddress. If the ISDN remote subaddress has been padded on the right with blanks, the length of that padding is not included in this total. :p If ISDN is not used, this field contains 0.	Display, ICF
196	Character	2	ISDN remote subaddress type (decimal). **00** NSAP. **01** User-specified.	Display, ICF
198	Character	40	ISDN remote subaddress (EBCDIC representation of the original hexadecimal value, padded on the right with zeros).	Display, ICF
238	Character	1	Reserved.	Display, ICF
239	Character	1	ISDN connection (decimal). **0** Incoming ISDN call. **1** Outgoing ISDN call. **Other** Non-ISDN connection.	Display, ICF
240	Binary	2	ISDN remote network address length in bytes. If the ISDN remote network address has been padded on the right with blanks, the length of that padding is not included. If ISDN is not used, this field contains 0.	Display, ICF
242	Character	32	The ISDN remote network address in EBCDIC, padded on the right with blanks, if necessary, to fill the field.	Display, ICF
274	Character	4	Reserved.	Display, ICF

Table 23. Get Attributes (continued)

Offset	Data Type	Length	Contents	File Type
278	Character	2	ISDN remote address extension length in bytes. Consists of the total of the lengths of the next two fields: ISDN remote address extension type and the ISDN remote address extension. If the ISDN remote address extension has been padded on the right with zeros, the length of that padding is not included. If ISDN is not used or there is no ISDN remote address extension, this field contains 0.	Display, ICF
280	Character	1	ISDN remote address extension type (decimal). **0** Address assigned according to ISO 8348/AD2 **2** Address not assigned according to ISO 8348/AD2 **Other** Reserved.	Display, ICF
281	Character	40	ISDN remote address extension (EBCDIC representation of the original hexadecimal value, padded on the right with zeros).	Display, ICF
321	Character	4	Reserved.	Display, ICF
325	Character	1	X.25 call type (decimal). **0** Incoming Switched Virtual Circuit (SVC) **1** Outgoing SVC **2** Not X.25 SVC **Other** Reserved.	Display, ICF

Note: The following information is available only for when your program was started as a result of a received program start request. Also, not all of the information will be available if the area to receive it is too small.

326	Character	64	Transaction program name. Name of the program specified to be started as a result of the received program start request, even if a routing list caused a different program to be started.	ICF
390	Binary	1	Length of the protected LUWID field. The valid values are 0 through 26.	ICF
391	Binary	1	Length of the qualified LU-NAME. The valid values are 0 through 17.	ICF
392	Character	17	Network qualified protected LU-NAME in the form: netid.luname. This field is blank if there is no network qualified protected LU-NAME.	ICF
409	Character	6	Protected LUWID instance number.	ICF
415	Binary	2	Protected LUWID sequence number.	ICF

Note: The following information is available only when a protected conversation is started on the remote system. That is, when a conversation is started with a SYNCLVL of *COMMIT. Also, not all of the information will be available if the area to receive it is too small.

417	Binary	1	Length of the unprotected LUWID field. The valid values are 0 through 26.	ICF
418	Binary	1	Length of the qualified LU-NAME. The valid values are 0 through 17.	ICF
419	Character	17	Network qualified unprotected LU-NAME in the form: netid.luname. This field is blank if there is no network qualified unprotected LU-NAME.	ICF
436	Character	6	Unprotected LUWID instance number.	ICF
442	Binary	2	Unprotected LUWID sequence number.	ICF

Table 23. Get Attributes (continued)

Offset	Data Type	Length	Contents	File Type

Note:

Type 1 Controllers available at V2R2 which support such things as windows and continued cursor progression.

Type 2 Controllers available at V2R3. These support all of the V2R2 functions as well as menu bars, continued-entry fields, edit masks, and simple hotspots.

Type 3 Controllers available at V3R1. These support all of the V2R2 and V2R3 functions. They also support text in the bottom border of windows.

Appendix B. Double-Byte Character Set Support

This appendix contains information that you need if you use double-byte characters. This includes the following topics:

- Double-byte character set (DBCS) fundamentals
- Processing double-byte characters
- Device file support
- Display support
- Copying files that contain double-byte characters
- Writing application programs that process double-byte characters
- DBCS font tables
- DBCS sort tables
- DBCS conversion dictionaries
- Using DBCS conversion

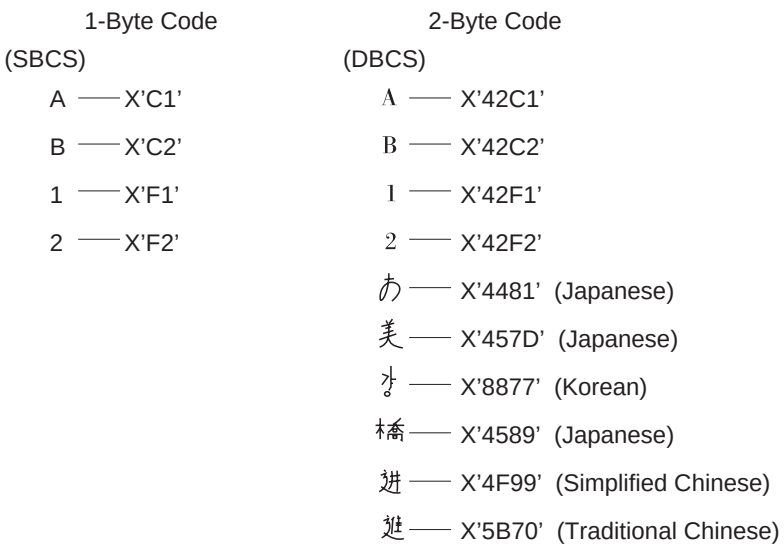
DBCS printer and spooling support information can be found in the *Printer Device Programming* book.

Double-Byte Character Set Fundamentals

Some languages, such as Chinese, Japanese, and Korean, have a writing scheme that uses many different characters that cannot be represented with single-byte codes. To create coded character sets for such languages, the system uses two bytes to represent each character. Characters that are encoded in two-byte code are called double-byte characters.

Figure 17 on page 184 shows alphanumeric characters coded in a single-byte code scheme and double-byte characters coded in a double-byte code scheme.

You can use double-byte characters as well as single-byte characters in one application. For instance, you may want to store double-byte data and single-byte data in your database, create your display screens with double-byte text and fields, or print reports with double-byte characters.



X'hhhh' indicates that the code has the hexadecimal value, "hhhh".

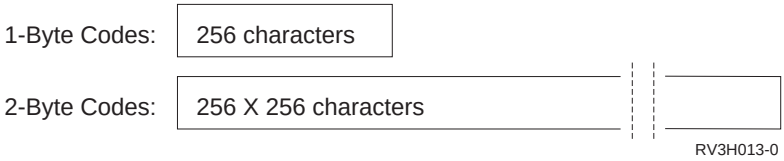


Figure 17. Single-byte and Double-byte Code Schemes

DBCS Code Scheme

IBM supports two DBCS code schemes: one for the host systems, the other for personal computers. The IBM-host code scheme has the following code-range characteristics:

First byte
hex 41 to hex FE

Second byte
hex 41 to hex FE

Double-byte blank
hex 4040

In the following figure, using the first byte as the vertical axis and the second byte as the horizontal axis, 256 x 256 intersections or **code points** are expressed. The lower-right code area is designated as the valid double-byte code area and x is assigned to the double-byte blank.

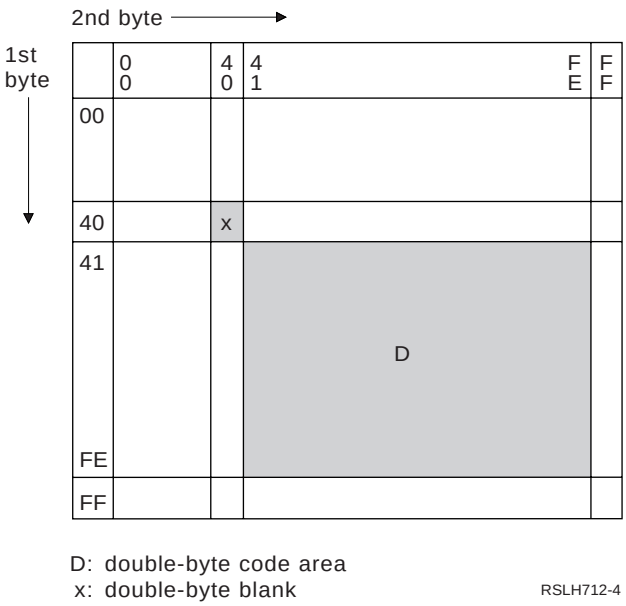


Figure 18. IBM-Host Code Scheme

By assigning the values hex 41 to hex FE in the first and second bytes as the DBCS codes, the codes can be grouped in wards with 192 code points in each ward. For example, the code group with the first byte starting with hex 42 is called *ward 42*. Ward 42 has the same alphanumeric characters as those in a corresponding single-byte EBCDIC code page, but with double-byte codes. For example, the character *A* is represented in single-byte EBCDIC code as hex C1 and in IBM-host code as hex 42C1.

The AS/400 system supports the following double-byte character sets:

- IBM Japanese Character Set
- IBM Korean Character Set
- IBM Simplified Chinese Character Set
- IBM Traditional Chinese Character Set

The following tables show the code ranges for each character set and the number of characters supported in each character set.

Table 24. IBM Japanese Character Set

Wards	Content	Number of Characters
40	Space in 4040	1
41 to 44	Non-Kanji characters • Greek, Russian, Roman numeric (Ward 41) • Alphanumeric and related symbols (Ward 42) • Katakana, Hiragana, and special symbols (Ward 43-44)	549
45 to 55	Basic Kanji characters	3226
56 to 68	Extended Kanji characters	3487

Table 24. IBM Japanese Character Set (continued)

Wards	Content	Number of Characters
69 to 7F	User-defined characters	Up to 4370
80 to FE	Reserved	

: Total number of IBM-defined characters: 7263

Table 25. IBM Korean Character Set

Wards	Content	Number of Characters
40	Space in 4040	1
41 to 46	Non-Hangeul/Hanja characters (Latin alphabet, Greek, Roman, Japanese Kana, numeric, special symbols)	939
47 to 4F	Reserved	
50 to 6C	Hanja characters	5265
6D to 83	Reserved	
84 to D3	Hangeul characters (Jamo included)	2672
D4 to DD	User-defined characters	Up to 1880
DE to FE	Reserved	

: Total number of IBM-defined characters: 8877

Table 26. IBM Simplified Chinese Character Set

Wards	Content	Number of Characters
40	Space in 4040	1
41 to 47	Non-Chinese characters (Latin alphabet, Greek, Russian, Japanese Kana, numeric, special symbols)	712
48 to 6F	Chinese characters: Level 1 and Level 2	3755 and 3008
70 to 75	Reserved	
76 to 7F	User-defined characters	Up to 1880
80 to FE	Reserved	

: Total number of IBM-defined characters: 7476

Table 27. IBM Traditional Chinese Character Set

Wards	Content	Number of Characters
40	Space in 4040	1

Table 27. IBM Traditional Chinese Character Set (continued)

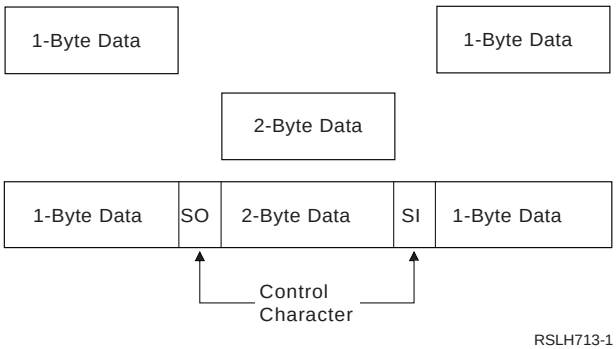
Wards	Content	Number of Characters
41 to 49	Non-Chinese characters (Latin alphabet, Greek, Roman, Japanese Kana, numeric, special symbols)	1003
4A to 4B	Reserved	
4C to 68	Primary Chinese characters	5402
69 to 91	Secondary Chinese characters	7654
92 to C1	Reserved	
C2 to E2	User-defined characters	Up to 6204
E3 to FE	Reserved	

: Total number of IBM-defined characters: 14060

This code scheme applies to the AS/400 system, System/36, System/38, as well as the System/370 system. A different DBCS code scheme, called the IBM Personal Computer DBCS code scheme, is used on the Personal System/55. For details of the IBM Personal Computer DBCS code scheme, refer to IBM PS/55 publications.

Shift-Control Characters

When the IBM-host code scheme is used, the system uses shift-control characters to identify the beginning and end of a string of double-byte characters. The shift-out (SO) character, hex 0E, indicates the beginning of a double-byte character string. The shift-in (SI) character, hex 0F, indicates the end of a double-byte character string.



Each shift-control character occupies the same amount of space as one alphanumeric character. By contrast, double-byte characters occupy the same amount of space as two alphanumeric characters.

When double-byte characters are stored in a graphic field or a variable of graphic data type, there is no need to use shift control characters to surround the double-byte characters.

Invalid Double-Byte Code and Undefined Double-Byte Code

Invalid double-byte code has a double-byte code value that is not in the valid double-byte code range. Figure 18 on page 185 shows valid double-byte code ranges. This is in contrast to undefined double-byte code where the double-byte code is valid, but no graphic symbol has been defined for the code.

Using Double-Byte Data

This section tells you where you can use double-byte data and discusses the limitations to its use.

Where You Can Use

You can use double-byte data in the following ways:

- As data in files:
  - Data in database files.
  - Data entered in input-capable and data displayed in output-capable fields of display files.
  - Data printed in output-capable fields in printer files.
  - Data used as literals in display files and printer files.
- As the text of messages.
- As the text of object descriptions.
- As literals and constants, and as data to be processed by high-level language programs.

Double-byte data can be displayed only at DBCS display stations and printed only on DBCS printers. Double-byte data can be written onto diskette, tape, disk, and optical storage.

Where You Cannot Use

You cannot use double-byte data in the following ways:

- As AS/400 object names.
- As command names or variable names in control language (CL) and other high-level languages.
- As displayed or printed output on alphanumeric work stations.

Double-Byte Character Size

When displayed or printed, double-byte characters usually are twice as wide as single-byte characters.

Consider the width of double-byte characters when you calculate the length of a double-byte data field because field lengths are usually identified as the number of single-byte character positions used. For more information on calculating the length of fields containing double-byte data, refer to the *DDS Reference*.

Processing Double-Byte Characters

Due to the large number of double-byte characters, the system needs more information to identify each double-byte character than is needed to identify each alphanumeric character.

There are two types of double-byte characters: basic and extended. These characters are usually processed by the device on which the characters are displayed or printed.

Basic Characters

Basic characters are frequently used double-byte characters that reside in the hardware of a DBCS-capable device. The number of double-byte characters stored in the device varies with the language supported and the storage size of the device. A DBCS-capable device can display or print basic characters without using the extended character processing function of the operating system.

Extended Characters

When processing extended characters, the device requires the assistance of the system. The system must tell the device what the character looks like before the device can display or print the character. **Extended characters** are stored in a DBCS font table, not in the DBCS-capable device. When displaying or printing extended characters, the device receives them from the DBCS font table under control of the operating system.

Extended character processing is a function of the operating system that is required to make characters stored in a DBCS font table available to a DBCS-capable device.

To request extended character processing, specify the double-byte extended character parameter, IGCEXNCHR(*YES), on the file creation command when you create a display (CRTDSPF command) or printer file (CRTPRTF command) that processes double-byte data. Because IGCEXNCHR(*YES) is the default value, the system automatically processes extended characters unless you instruct it otherwise. You can change this file attribute by using a change file (CHGDSPF or CHGPRTF) or override file (OVRDSPF or OVRPRTF) command. For example, to override the display file DBCSDSPF so that extended characters are processed, enter:

```
OVRDSPF DSPF(DBCSDSPF) IGCEXNCHR(*YES)
```

Notes:

1. The system ignores the IGCEXNCHR parameter when processing alphanumeric files.
2. When you use the Japanese 5583 Printer to print extended characters, you must use the Kanji print function of the Advanced DBCS Printer Support for AS/400 licensed program. Refer to the *Kanji Print Function User's Guide and Reference* for how to use this utility.

What Happens When Extended Characters Are Not Processed

When extended characters are not processed, the following happens:

- Basic double-byte characters are displayed and printed.

- On displays, the system displays the undefined character where it would otherwise display extended characters.
- On printed output, the system prints the undefined character where it would otherwise print extended characters.
- The extended characters, though not displayed or printed, are stored correctly in the system.

Device File Support

The following sections describe DBCS-capable device files and special considerations for working with DBCS-capable device files. Data description specifications (DDS), a language used to describe files, can be used with DBCS-capable device files. For information about using DDS, refer to the *DDS Reference* .

What a DBCS File Is

A **DBCS file** is a file that contains double-byte data or is used to process double-byte data. Other files are called **alphanumeric files**.

The following types of device files can be DBCS files:

- Display
- Printer
- Tape
- Diskette
- ICF

When to Indicate a DBCS File

You should indicate that a file is DBCS in one or more of the following situations:

- The file receives input, or displays or prints output, which has double-byte characters.
- The file contains double-byte literals.
- The file has double-byte literals in the DDS that are used in the file at processing time (such as constant fields and error messages).
- The DDS of the file includes DBCS keywords. See the *DDS Reference* for information on these keywords.
- The file stores double-byte data (database files).

How to Indicate a DBCS File

You must indicate that a device file is a DBCS file in order for the system to process double-byte data properly. You can do this in one of the following ways:

- Through DDS
  - DDS provides fields of the following data types.
    - **DBCS-only fields:** display and accept only double-byte characters. Double-byte characters in a DBCS-only field are enclosed in shift-out and shift-in characters that have to be paired.

- **DBCS-open fields:** display and accept both single-byte and double-byte characters. Double-byte characters are enclosed in shift-out and shift-in characters that have to be paired.
- **DBCS-either fields:** display and accept *either* single-byte or double-byte characters, but not *both*. Double-byte characters are enclosed in shift-out and shift-in character pairs.
- **DBCS-graphic fields:** display and accept only double-byte characters. Characters in a DBCS-graphic field do not have shift-out and shift-in characters. The AS/400 DBCS-graphic field is equivalent to a System/370 DBCS field.

– In ICF files, by defining fields with DBCS-open data type (type O).

– In printer files, by defining fields with DBCS-open data type (type O) and DBCS-graphic data type (type G).

– In display files, by defining fields with DBCS-only data type (type J), DBCS-either data type (type E), DBCS-open data type (type O), or DBCS-graphic data type (type G).

– By using a double-byte literal that is used with the file at processing time, such as literals specified with the Default (DFT) and Error Message (ERRMSG) DDS keywords.

Note: You may also use double-byte literals as text and comments in a file, such as with the DDS keyword TEXT. However, the system does not consider a file, whose only DBCS usage is that it has double-byte comments, to be a DBCS file.

– By specifying the Alternative Data Type (IGCALTTYP) DDS keyword in display and printer files. This keyword lets you use display and printer files with both alphanumeric and double-byte applications. When you put the IGCALTTYP keyword into effect, you can use double-byte data with the file.

Put the IGCALTTYP keyword into effect by creating, changing, or overriding display and printer files with the IGCDTA(*YES) value. You can put the IGCALTTYP keyword into effect for display and printer files by specifying IGCDTA(*YES) on the following device file commands:

- Create Display File (CRTDSPF)
- Create Printer File (CRTPRTF)
- Change Display File (CHGDSPF)
- Change Printer File (CHGPRTF)
- Override with Display File (OVRDSPF)
- Override with Printer File (OVRPRTF)

When you specify IGCDTA(*NO), the IGCALTTYP keyword is not in effect and you can use only alphanumeric data with the file. Changing or overriding the file to put the IGCALTTYP keyword into effect does not change the DDS of the file.

Except when using the IGCALTTYP function, you do not need to specify IGCDTA(*YES) on the file creation command if you have already specified DBCS functions in the DDS. Instead, specify IGCDTA(*YES) when the file has DBCS functions that are not indicated in the DDS. For example, specify IGCDTA(*YES) on the file creation command if the file is intended to contain double-byte data.

• By specifying IGCDTA(*YES) on the following device file creation commands:

– Create Diskette File (CRTDKTF)

  - Create Display File (CRTDSPF)
  - Create Printer File (CRTPRTF)
  - Create Tape File (CRTTAPF)
- By specifying IGCDTA(*YES) on the following database file creation commands:
  - Create Physical File (CRTPF)
  - Create Source Physical File (CRTSRCPF)

Improperly Indicated DBCS Files

If you do not properly indicate that a file is a DBCS file, one of the following happens:

- For printer files, printer data management assumes the output data to the printer does not contain double-byte data. The end result depends on the type of printer the data is printed on and the status of the replace unprintable character parameter for the printer file you are using.

  If the replace-unprintable-character option is selected, printer data management interprets shift-control characters as unprintable characters and replaces them with blanks. The double-byte data itself is interpreted as alphanumeric data, and the printer attempts to print it as such. The printed double-byte data does not make sense.

  If the replace-unprintable-character option is not selected and the printer is an alphanumeric printer, the double-byte data, including the control characters, is sent as is to the printer. On most alphanumeric printers, the shift-control characters are not supported, and an error will occur at the printer.

  If the replace-unprintable-character option is not selected and the printer is a DBCS printer, the double-byte data is printed with the exception of extended characters. Because the file was not indicated as a DBCS file, the system will not perform extended character processing. The extended characters are printed with the symbol for undefined double-byte characters.
- For display files, display data management assumes that the output data to the display does not contain double-byte data. The end result depends on whether the display is an alphanumeric or DBCS display.

  If the display is an alphanumeric display, the double-byte data is interpreted as alphanumeric data. The shift-control characters appear as anks. The displayed double-byte data does not make sense.

  If the display is a DBCS display, the double-byte data is displayed with the exception of extended characters. The system does not perform extended character processing on the data. Therefore, extended characters are displayed with the symbol for undefined double-byte characters.
- The system does not recognize literals with DBCS text as double-byte literals if the source file is not specified as a DBCS file.

Making Printer Files Capable of DBCS

In many cases, printer files are used by the system to produce data that will eventually be printed or displayed. In these cases, the data is first placed into a spooled file using one of the IBM-supplied printer files. The data is then taken from the spooled file and is displayed or printed based on the request of the user.

When the data involved contains double-byte characters, the printer file that is used to place the data into the spooled file must be capable of processing double-byte data. A printer file is capable of processing double-byte data when *YES is specified on the IGCDTA parameter for the file. In most cases, the system recognizes the

occurrence of double-byte data and takes appropriate measures to ensure the printer file that is used is capable of processing double-byte data.

In some cases, however, the system cannot recognize the occurrence of double-byte data and may attempt to use a printer file that is not capable of processing double-byte data. If this occurs, the output at the display or printer may not be readable. This can happen when object descriptions containing double-byte characters are to be displayed or printed on an alphanumeric device.

To ensure that you receive correct results when you display or print double-byte characters, some recommendations should be followed. Action is required on your part if you have a single-byte national language installed as a secondary language. Printer files that are received as part of the DBCS version of a product are always capable of processing DBCS data.

The following recommended actions should be performed after the product or feature has been installed:

1. If all printers and display devices attached to your system are DBCS-capable, you can enable all printer files for double-byte data. For IBM-supplied printer files that are received as part of a single-byte secondary language feature, you can enable all printer files by issuing the following command:

   ```
   CHGPRTF FILE(*ALL/*ALL) IGCDTA(*YES)
   ```

   After this command has been completed, all printer files in all libraries will be enabled for double-byte data. The change will be a permanent change.
2. If all printer and display devices attached to your system are not DBCS-capable, it is recommended that you do not enable all IBM-supplied printer files.

   Instead, use the library search capabilities of the system to control which printer files will be used for any particular job. When the potential exists that double-byte data will be encountered, the library list for the job should be such that the printer files that are DBCS-enabled will be found first in the library list. Conversely, if only single-byte data is expected to be encountered, the library list should be set up so the printer files that are not enabled for DBCS will be found first. In this way, the printer file capabilities will match the type of data that will be processed. The decision as to what type of printer file to use is made on the basis of what type of data will be processed. The device that will be used to actually display or print the data may also influence this decision.

   In some cases it may be desirable to make the printer file only temporarily DBCS-capable instead of making a permanent change. For a specific job, you can make this temporary change by using the OVRPRTF command.

   To temporarily enable a specific printer file, you can use the following command:

   ```
   OVRPRTF FILE(filename) IGCDTA(*YES)
   ```

   Where filename is the name of the printer file you want to enable.

Display Support

The following sections describe information on displaying double-byte characters.

Inserting Shift-Control Characters

The system inserts shift-control characters into DBCS-only fields automatically.

To insert shift-control characters into open fields or either fields, do the following:

1. Position the cursor in the field in which you want to insert double-byte data.
2. Press the Insert Shift Control Character key (according to your DBCS display station user's guide).

The system inserts a pair of shift-control characters at the same time, as follows (where 0_E represents the shift-out character and 0_F represents the shift-in character):

0_E0_F

The system leaves the cursor under the shift-in character and puts the keyboard in insert mode. Insert double-byte characters between the shift-control characters. To insert double-byte characters, start keying in double-byte characters at the cursor position. For example, enter the double-byte character string D1D2D3, as follows (where 0_E represents the shift-out character, 0_F represents the shift-in character, and D1, D2, and D3 represent three double-byte characters):

$0_ED1D2D30_F$

To find out if a field already has the shift-control characters, press the Display Shift Control Character key.

DBCS-graphic fields store double-byte characters without requiring the use of shift control characters. Shift control characters should not be inserted in graphic fields.

Number of Displayed Extended Characters

The system can display up to 512 different extended characters on a Japanese display at one time. Additional extended characters are displayed as undefined characters. However, the additional extended characters are stored correctly in the system.

Number of Input Fields on a Display

The use of DBCS input fields affects the total number of input fields allowed on a display. For a local 5250 display station, you can specify as many as 256 input fields. However, each three instances of a DBCS field reduces the maximum number of fields by one. For example, if there are 9 DBCS fields on a display, then the maximum is 256 − (9/3) = 253 input fields.

Effects of Displaying Double-Byte Data at Alphanumeric Work Stations

Alphanumeric display stations cannot display double-byte data correctly. If you try to display double-byte data at an alphanumeric display station, the following happens:

- The system sends an inquiry message to that display station, asking whether you want to continue using the program with double-byte data or to cancel it.
- If you continue using the program, the system ignores the shift-control characters and interprets the double-byte characters as though they were single-byte characters. Displayed double-byte data does not make sense.

Copying Files

You can copy both spooled and nonspooled DBCS files.

Copying Spooled Files

Copy spooled files that have double-byte data by using the Copy Spooled File (CPYSPLF) command. However, the database file to which the file is being copied must have been created with the IGCDTA(*YES) value specified.

When copying spooled files to a database file that contains double-byte data, an extra column is reserved for the shift-out character. This shift-out character is placed between the control information for the record and the user data. The following table shows the shift-out character column number, based on the value specified for the Control Character (CTLCHAR) keyword:

CTLCHAR Value	Column for Shift-Out Character
*NONE	1
*FCFC	2
*PRTCTL	5
*S36FMT	10

Copying Nonspooled Files

You can use the Copy File (CPYF) command to copy double-byte data from one file to another.

When copying data from a double-byte database file to an alphanumeric database file, specify one of the following on the CPYF command:

- If both files are source files or if both files are database files, you can specify either the FMTOPT(*MAP) parameter or the FMTOPT(*NOCHK) parameter.
- If one file is a source file and the other file is a database file, specify the FMT(*CVTSRC) parameter.

When you copy DBCS files to alphanumeric files, the system sends you an informational message describing the difference in file types.

Either the FMTOPT(*MAP) or FMTOPT(*NOCHK) option of the copy file function must be specified for copies from a physical or logical file to a physical file when there are fields with the same name in the from-file and to-file, but the data type for fields is as shown in the following table.

From-File Field Data Type	To-File Field Data Type
A (character)	J (DBCS-only)
O (DBCS-open)	J (DBCS-only)
O (DBCS-open)	E (DBCS-either)
E (DBCS-either)	J (DBCS-only)
J (DBCS-only)	G (DBCS-graphic)
O (DBCS-open)	G (DBCS-graphic)
E (DBCS-either)	G (DBCS-graphic)
G (DBCS-graphic)	J (DBCS-only)
G (DBCS-graphic)	O (DBCS-open)
G (DBCS-graphic)	E (DBCS-either)
G (UCS-2 graphic)	A (Character (CCSID non-65535))

From-File Field Data Type	To-File Field Data Type
G (UCS-2 graphic)	J (DBCS-open (CCSID non-65535))
G (UCS-2 graphic)	E (DBCS-either (CCSID non-65535))
G (UCS-2 graphic)	J (DBCS-only (CCSID non-65535))
G (UCS-2 graphic)	G (DBCS-graphic)
A (Character (CCSID non-65535))	G (UCS-2 graphic)
O (DBCS-open (CCSID non-65535))	G (UCS-2 graphic)
E (DBCS-either (CCSID non-65535))	G (UCS-2 graphic)
J (DBCS-only (CCSID non-65535))	G (UCS-2 graphic)
G (DBCS-graphic)	G (UCS-2 graphic)

When you use FMTOPT(*MAP) on the CPYF command to copy data to a DBCS-only field or DBCS-graphic field, the corresponding field in the from-file must *not* be:

- Less than a 2-byte character field
- An odd-byte-length character field
- An odd-byte-length DBCS-open field

Note: DBCS-graphic is the only type allowed to be CCSID 65535 when using FMTOPT(*MAP) copying from or to a UCS-2 graphic field. UCS-2 graphic *cannot be* CCSID 65535.

If you attempt to copy with one of these specified in the from-field, an error message is sent.

When you copy double-byte data from one database file to another with the FMTOPT(*MAP) parameter specified, double-byte data will be copied correctly. The system will perform correct padding and truncation of double-byte data to ensure data integrity.

When using the CPYF command with FMTOPT(*MAP) to copy a DBCS-open field to a graphic field, a conversion error occurs if the DBCS-open field contains any SBCS data (including blanks).

Application Program Considerations

The following sections describe considerations for writing applications that process double-byte data.

Designing Application Programs That Process Double-Byte Data

Design your application programs for processing double-byte data in the same way you design application programs for processing alphanumeric data, with the following additional considerations:

- Identify double-byte data used in the database files.
- Design display and printer formats that can be used with double-byte data.
- If needed, provide DBCS conversion as a means of entering double-byte data for interactive applications. Use the DDS keyword for DBCS conversion (IGCCNV) to specify DBCS conversion in display files. Because DBCS work stations provide a variety of double-byte data entry methods, you are not required to use the AS/400 DBCS conversion function to enter double-byte data.
- Create double-byte messages to be used by the program.

- Specify extended character processing so that the system prints and displays all double-byte data. See “Extended Characters” on page 189 for instructions.
- Determine whether additional double-byte characters need to be defined. User-defined characters can be defined and maintained using the **character generator utility (CGU)**. Information on CGU can be found in the *ADTS/400: Character Generator Utility* book.

When you write application programs to process double-byte data, make sure that the double-byte data is always processed in a double-byte unit and do not split a double-byte character.

Changing Alphanumeric Application Programs to DBCS Application Programs

If an alphanumeric application program uses externally described files, you can change that application program to a DBCS application program by changing the externally described files. To convert an application program, do the following:

1. Create a duplicate copy of the source statements for the alphanumeric file that you want to change.
2. Change alphanumeric constants and literals to double-byte constants and literals.
3. Change fields in the file to the open (O) data type or specify the Alternative Data Type (IGCALTTYP) DDS keyword so that you can enter both double-byte and alphanumeric data in these fields. You may want to change the length of the fields as the double-byte data takes more space.
4. Store the converted file in a separate library. Give the file the same name as its alphanumeric version.
5. When you want to use the changed file in a job, change the library list, using the Change Library List (CHGLIBL) command, for the job in which the file will be used. The library in which the DBCS display file is stored is then checked before the library in which the alphanumeric version of the file is stored.

DBCS Font Tables

DBCS font tables contain the images of the double-byte extended characters used on the system. The system uses these images to display and print extended characters.

The following DBCS font tables are objects that you can save or restore. These font tables are distributed with the DBCS national language versions of the OS/400 licensed program:

QIGC2424
: A Japanese DBCS font table used to display and print extended characters in a 24-by-24 dot matrix image. The system uses the table with Japanese display stations, printers attached to display stations, 5227 Model 1 Printer, and the 5327 Model 1 Printer.

QIGC2424C
: A Traditional Chinese DBCS font table used to print extended characters in a 24-by-24 dot matrix image. The system uses the table with the 5227 Model 3 Printer and the 5327 Model 3 Printer.

QIGC2424K
A Korean DBCS font table used to print extended characters in a 24-by-24 dot matrix image. The system uses the table with the 5227 Model 2 Printer and the 5327 Model 2 Printer.

QIGC2424S
A Simplified Chinese DBCS font table used to print extended characters in a 24-by-24 dot matrix image. The system uses the table with the 5227 Model 5 Printer.

QIGC3232
A Japanese DBCS font table used to print characters in a 32-by-32 dot matrix image. The system uses the table with the 5583 Printer and the 5337 Model 1 Printer.

QIGC3232S
A Simplified Chinese DBCS font table used to print characters in a 32-by-32 dot matrix image. The system uses the table with the 5337 Model R05 Printer.

All DBCS font tables have an object type of *ICGTBL. You can find instructions for adding user-defined characters to DBCS font tables in the *ADTS/400: Character Generator Utility* book.

Commands for DBCS Font Tables

The following commands allow you to manage and use DBCS font tables:

- Check DBCS Font Table (CHKIGCTBL)
- Copy DBCS Font Table (CPYIGCTBL)
- Delete DBCS Font Table (DLTIGCTBL)
- Start Character Generator Utility (STRCGU)
- Start Font Management Aid (STRFMA)

Finding Out if a DBCS Font Table Exists

Use the Check DBCS Font Table (CHKIGCTBL) command to find out if a DBCS font table exists in your system.

For example, to find out if the table QIGC2424 exists, enter:

```
CHKIGCTBL IGCTBL(QIGC2424)
```

If the table does not exist, the system responds with a message. If the table does exist, the system simply returns without a message.

Check for the existence of a table when adding a new type of DBCS work station to make sure that the table used by the device exists in the system.

Copying a DBCS Font Table onto Tape or Diskette

Use the Copy DBCS Font Table (CPYIGCTBL) command to copy a DBCS font table onto tape or diskette.

The DBCS font tables are saved when you use the Save System (SAVSYS) command so you do not have to use the CPYIGCTBL command when performing normal system backup.

When to Copy a Table onto Tape or Diskette

Copy a DBCS font table onto tape or diskette in the following instances:

- Before deleting that table.
- After new user-defined characters are added to the tables.
- When planning to use the tables on another system.

How to Copy a Table onto Tape or Diskette

To copy a DBCS font table onto a tape or diskettes, do the following:

1. Make sure that you have a tape or diskettes initialized to the *DATA format. If necessary, initialize the tape or diskettes by specifying the FMT(*DATA) parameter on the Initialize Diskette (INZDKT) command. See the *Tape and Diskette Device Programming* book for complete instructions on initializing tapes and diskettes.
2. Load the initialized tape or diskette onto the system.
3. Enter the CPYIGCTBL command as follows:
   a. Choose the value OPTION(*OUT).
   b. Use the DEV parameter to select the device to which you want to copy the table.
   c. Use the SELECT and RANGE parameters to specify which portion of the table you want copied from the system. See the description of the CPYIGCTBL command in the *CL Reference (Abridged)*for instructions on choosing SELECT and RANGE parameter values.

   The following are two examples of the CPYIGCTBL command used to copy a DBCS font table to removable media.

   - To copy the DBCS font table QIGC2424 onto diskettes, enter:

     ```
     CPYIGCTBL IGCTBL(QIGC2424) OPTION(*OUT)  +
       DEV(QDKT)
     ```

   - To copy just the user-defined characters from DBCS font table QIGC2424 onto tape, enter:

     ```
     CPYIGCTBL IGCTBL(QIGC2424) OPTION(*OUT)  +
       DEV(QTAP01) SELECT(*USER)
     ```

4. Press the Enter key. The system copies the DBCS font table onto the specified media.
5. Remove the tape or diskette after the system finishes copying the table.

Copying a DBCS Font Table from Tape or Diskette

Use the Copy DBCS Font Table (CPYIGCTBL) command to copy a DBCS font table from a tape or a diskette onto the system. The system automatically creates the DBCS font table again when copying its contents if the following are true:

- The specified table does not already exist in the system.
- The media from which you are copying the table contains all of the IBM-defined double-byte characters.
- SELECT(*ALL) or SELECT(*SYS) is specified on the CPYIGCTBL command.

How to Copy a Table from a Tape or Diskette

To copy a DBCS font table from tape or diskette onto the system:

1. Load the removable media from which the table will be copied onto the system.

2. Enter the CPYIGCTBL command as follows:
   a. Choose the OPTION(*IN) value.
   b. Use the DEV parameter to select the device from which to copy the DBCS font table.
   c. Use the SELECT and RANGE parameters to specify which portion of the table will be copied from the tape or diskette. See the *CL Reference (Abridged)* for a description of the CPYIGCTBL command and for instructions on choosing SELECT and RANGE parameter values.

   The following are two examples of commands used to copy a DBCS font table to the system.

   - To copy the DBCS font table QIGC2424 from diskette, enter:

     ```
     CPYIGCTBL IGCTBL(QIGC2424) OPTION(*IN) +
       DEV(QDKT)
     ```

   - To copy just the user-defined characters from DBCS font table QIGC2424 from tape and to replace the user-defined characters in the table with the ones from the tape, enter:

     ```
     CPYIGCTBL IGCTBL(QIGC2424) OPTION(*IN) +
       DEV(QTAP01) SELECT(*USER) RPLIMG(*YES)
     ```

3. Press the Enter key. The system copies the DBCS font table from the tape or diskette onto the system.
4. Remove the tape or diskette after the system finishes copying the table.

Deleting a DBCS Font Table

Use the Delete DBCS Font Table (DLTIGCTBL) command to delete a DBCS font table from the system.

When to Delete a DBCS Font Table

Delete an unused DBCS font table to free storage space. For example, if you do not plan to use Japanese printer 5583 or 5337 with your system, font table QIGC3232 is not needed and can be deleted.

How to Delete a DBCS Font Table

When deleting a table, do the following:

1. If desired, copy the table onto tape or diskettes. See “Copying a DBCS Font Table onto Tape or Diskette” on page 198 for instructions. If you do not copy the table to removable media before deleting it, you will not have a copy of the table for future use.
2. Vary off all devices using that table.
3. Enter the DLTIGCTBL command.

   For example, to delete the DBCS font table QIGC3232, enter:

   ```
   DLTIGCTBL IGCTBL(QIGC3232)
   ```

4. Press the Enter key. The system sends inquiry message CPA8424 to the system operator message queue for you to confirm your intention to delete a DBCS table.
5. Respond to the inquiry message. The system sends you a message when it has deleted the table.

Note: Do not delete a DBCS font table if any device using that table is currently varied on. Also, make sure that the affected controller is not varied on. If you

try to delete the table while the device and controller are varied on, the system reports any devices attached to the same controller(s) as those devices, and the controller(s) as damaged the next time you try to print or display extended characters on an affected device. If such damage is reported, do the following:

1. Vary off the affected devices, using the Vary Configuration (VRYCFG) command.
2. Vary off the affected controller.
3. Vary on the affected controller.
4. Vary on the affected devices.
5. Continue normal work.

Starting the Character Generator Utility

Use the STRCGU command to start the character generator utility. You may call the CGU main menu or specify a specific CGU function, depending on the parameter used. Refer to the *ADTS/400: Character Generator Utility* book for more information.

Copying User-Defined Double-Byte Characters

Use the STRFMA command to copy user-defined double-byte characters between an AS/400 DBCS font table and a user font file at a Personal System/55, a 5295 Display Station, or an InfoWindow 3477 Display Station. Refer to the *Font Management Aid User's Guide* on how to use this command.

DBCS Font Files

In addition to the system-supplied DBCS font tables, the system also provides DBCS font files. These DBCS font files are physical files which contain frequently used double-byte characters. When using the character generator utility, you can use the characters in these files as the base for a new user-defined character. These files are supplied with read-only authority as they are not to be changed. If you do not use character generator utility or theAdvanced DBCS Printer Support for AS/400 licensed program, you may delete these files to save space. They all exist in the QSYS library.

The following DBCS font files are distributed with the DBCS national language versions of the OS/400 licensed program. They are used as a reference for the CGU and the AS/400 Advanced DBCS Printer Support for AS/400 licensed program.

QCGF2424
: A Japanese DBCS font file used to store a copy of the Japanese DBCS basic character images.

QCGF2424K
: A Korean DBCS font file used to store a copy of the Korean DBCS basic character images.

QCGF2424C
: A Traditional Chinese DBCS font file used to store a copy of the Traditional Chinese DBCS basic character images.

QCGF2424S
A Simplified Chinese DBCS font file used to store a copy of the Simplified Chinese DBCS basic character images.

DBCS Sort Tables

DBCS sort tables contain the sort information and collating sequences of all the double-byte characters used on the system. The system uses these tables to sort double-byte characters using the sort utility.

DBCS sort tables are objects that you can save, restore and delete. Using the character generator utility you can also add, delete and change entries in these tables corresponding to the image entries in the DBCS font tables. For Japanese use only, you can also copy the DBCS master sort table to and from a data file.

The following DBCS sort tables are distributed with the DBCS national language versions of OS/400 licensed program:

QCGMSTR
A Japanese DBCS master sort table used to store the sort information for the Japanese double-byte character set.

QCGACTV
A Japanese DBCS active sort table used to store the sort collating sequences for the Japanese double-byte character set.

QCGMSTRC
A Traditional Chinese DBCS master sort table used to store the sort information for the Traditional Chinese double-byte character set.

QCGACTVC
A Traditional Chinese DBCS active sort table used to store the sort collating sequences for the Traditional Chinese double-byte character set.

QCGACTVK
A Korean DBCS active sort table used to map Hanja characters to Hangeul characters with equivalent pronunciation.

QCGMSTRS
A Simplified Chinese DBCS master sort table used to store the sort information for the Simplified Chinese double-byte character set.

QCGACTVS
A Simplified Chinese DBCS active sort table used to store the sort collating sequences for the Simplified Chinese double-byte character set.

You may sort Japanese, Korean, Simplified Chinese, and Traditional Chinese double-byte characters. Each of these languages have two DBCS sort tables, a DBCS master sort table and a DBCS active sort table, except for Korean which has only a DBCS active sort table. The DBCS master sort table contains sort information for all defined DBCS characters. The DBCS active sort table for Japanese, Simplified Chinese, and Traditional Chinese is created from the master sort table information and contains the collating sequences for the double-byte characters of that given language. These collating sequences have a purpose similar to the EBCDIC and ASCII collating sequences for the single-byte alphanumeric character set. For Korean characters, the Hangeul characters are assigned both their collating sequence as well as their DBCS codes according to their pronunciation. Hence, a separate collating sequence is not required, and each

of the Hanja characters is mapped to a Hangeul character of the same pronunciation using the DBCS active sort table QCGACTVK.

All DBCS sort tables have an object type of *IGCSRT.

Commands for DBCS Sort Tables

The following commands allow you to manage and use DBCS sort tables.

- Check Object (CHKOBJ)
- Save Object (SAVOBJ)
- Restore Object (RSTOBJ)
- Copy DBCS Sort Table (CPYIGCSRT) (for Japanese table only)
- Delete DBCS Sort Table (DLTIGCSRT)
- Start Character Generator Utility (STRCGU) (information on CGU can be found in the *ADTS/400: Character Generator Utility* book.)

Using DBCS Sort Tables on the System

You can save the tables to tape or diskette, delete them from the system, and restore them to the system. The Japanese DBCS master sort table can also be copied to a data file and copied from a data file so that it can be shared with a System/36 or Application System/Entry (AS/Entry) system. You can also add sort information for each user-defined character, and add that character to the DBCS collating sequence, as you create it using the character generator utility.

Finding Out if a DBCS Sort Table Exists

Use the Check Object (CHKOBJ) command to find out if a DBCS sort table exists in your system.

For example, to find out if the table QCGMSTR exists, enter:

```
CHKOBJ OBJ(QSYS/QCGMSTR) OBJTYPE(*IGCSRT)
```

If the table does not exist, the system responds with a message. If the table does exist, the system simply returns without a message.

Check for the existence of a DBCS active sort table when you want to sort double-byte characters for the first time. The DBCS active table for the DBCS language must exist to sort the characters.

Saving a DBCS Sort Table onto Tape or Diskette

Use the Save Object (SAVOBJ) command to save a DBCS sort table onto tape or diskette. Specify *IGCSRT for the object type.

The DBCS sort tables are saved when you use the SAVSYS command so you do not have to use the SAVOBJ command when performing normal system backup.

When to Save a DBCS Sort Table onto Tape or Diskette

Save a DBCS sort table onto tape or diskette in the following instances:

- Before deleting that table

- After information is added, updated, or changed in the tables using the character generator utility
- When planning to use the tables on another AS/400 system

Restoring a DBCS Sort Table from Tape or Diskette

Use the RSTOBJ command to restore a DBCS sort table from a tape or a diskette onto the system. The tables on the tape or diskette must previously have been saved using the SAVOBJ command. Specify *IGCSRT for the object type. The system automatically re-creates the DBCS sort table when the specified table does not already exist in the system.

These tables must be restored to the QSYS library for the system to know they exist. For that reason, RSTOBJ restores *IGCSRT objects only to the QSYS library and only if the objects do not already exist there.

Copying a Japanese DBCS Master Sort Table to a Data File

Through the character generator utility, use the CPYIGCSRT command to copy the Japanese DBCS master sort table (QCGMSTR) to a data file. This data file can then be moved to a System/36 system or AS/Entry system to replace the Japanese master sort table there.

When to Copy the Japanese DBCS Master Sort Table to a Data File

Copy the Japanese DBCS master sort table to a data file in the following instances:

- When planning to move the table to the System/36 or AS/Entry for use there. You should always transport the Japanese DBCS master sort table together with the Japanese DBCS font tables.
- Before deleting that table, as an alternative to the SAVOBJ command. You can then keep the file or save it on diskette or tape.

How to Copy the Japanese DBCS Master Sort Table to a Data File

Note: In this section, the AS/Entry system also applies to every instance of System/36.

To copy the Japanese DBCS master sort table to a data file, do the following.

1. Decide what data file you want to copy it to. The file need not exist, it will be automatically created.
2. Enter the CPYIGCSRT command as follows:
   a. Choose the value OPTION(*OUT).
   b. Use the FILE parameter to specify the name of the data file to which you want to copy the master table. If you are transporting the master table to the System/36 for use there, you should specify a file name of #KAMAST, or you will have to rename the file when you get it to the System/36. Use the AS/400 CPYF command for copying the file onto diskette, and the System/36 TRANSFER command for copying the file from diskette to the System/36.

c. Use the MBR parameter to specify the name of the data file member to which you want to copy the master table. If you are transporting the master table to the System/36 for use there, you should specify *FILE for the MBR parameter.

3. Press the Enter key. The system creates the file and member if they do not exist, and overwrites the existing member if they do exist.
4. If you now transport this file to your System/36 to replace the #KAMAST file there, you should also use the SRTXBLD procedure to update the active table to reflect the new master table.

Copying a Japanese DBCS Master Sort Table from a Data File

Use the CPYIGCSRT command to copy the Japanese DBCS master sort table (QCGMSTR) from a data file.

When to Copy the Japanese DBCS Master Sort Table from a Data File

You may use the *System/36 Migration Planning* book to migrate the System/36 or AS/Entry master sort file (#KAMAST) to the AS/400 system. When you migrate the #KAMAST file using the *System/36 Migration Planning* book, you do not have to use the CPYIGCSRT command.

Copy the Japanese DBCS master sort table from a data file in the following instances:

- When you do not use the *System/36 Migration Planning*book, you may copy the #KAMAST file from the System/36 or AS/Entry to the AS/400 system. Then use the CPYIGCSRT command to copy sort information from the #KAMAST file to the AS/400 master sort table (QCGMSTR). Delete the #KAMAST file from the AS/400 system after you complete the copy operation.
- When you have copied a version of the master table to a data file and you now want to restore that version.

You should always migrate or copy the Japanese DBCS master sort table together with the Japanese DBCS font tables.

How to Copy the Japanese DBCS Master Sort Table from a Data File

To copy the Japanese DBCS master sort table from a data file, do the following:

1. Enter the CPYIGCSRT command as follows:
   a. Choose the value OPTION(*IN).
   b. Use the FILE parameter to specify the name of the data file that contains a migrated System/36 or AS/Entry master file or an AS/400 master table previously copied to the file using OPTION(*OUT) with the CPYIGCSRT command. To migrate your System/36 or AS/Entry master file without using the *System/36 Migration Planning* book, use the TRANSFER command with the IFORMAT parameter on the System/36 or AS/Entry to save the #KAMAST master file on diskette. Use the AS/400 Copy File (CPYF) command to copy the master file #KAMAST from diskette. Use the CPYIGCSRT command as described here to copy data from the file to the AS/400 Japanese DBCS master sort table.
   c. Use the MBR parameter to specify the name of the data file member from which you want to copy the master table data.

2. Press the Enter key. Even though the information in the existing Japanese DBCS master sort table is overridden, that table must exist before you can use this command.
3. To update the Japanese DBCS active table to reflect the new copied information, use the SRTXBLD procedure in the System/36 or AS/Entry environment, or the STRCGU command specifying OPTION(5). This must be done before you can use the sort utility to sort Japanese double-byte characters.

Deleting a DBCS Sort Table

Use the DLTIGCSRT command to delete a DBCS sort table from the system.

When to Delete a DBCS Sort Table

Delete an unused DBCS sort table to free disk space, but you should always first save a copy of the table using the SAVOBJ command. You should delete the DBCS master sort table for a DBCS language if any of the following are true:

1. You will not be creating any new characters for that language using the character generator utility.
2. You will not be using the sort utility to sort characters for that language.

You should delete the DBCS active sort table for a DBCS language if you will not be using the sort utility to sort characters for that language. The DBCS active sort table must be on the system to use the sort utility for this language.

How to Delete a DBCS Sort Table

When deleting a table, do the following:

1. If desired, save the table onto tape or diskettes. See “Saving a DBCS Sort Table onto Tape or Diskette” on page 203 for instructions. If you do not save the table onto removable media before deleting it, you will not have a copy of the table for future use.
2. Enter the DLTIGCSRT command.

   For example, to delete the DBCS sort table QCGACTV, enter:

   ```
   DLTIGCSRT IGCSRT(QCGACTV)
   ```
3. Press the Enter key. The system sends you a message when it has deleted the table.

DBCS Conversion Dictionaries

The DBCS conversion dictionary is a collection of alphanumeric entries and their related DBCS words. The system refers to the dictionary when performing DBCS conversion. See “How DBCS Conversion Works” on page 215 for information on how the system uses the DBCS conversion dictionary during DBCS conversion.

All DBCS conversion dictionaries have an object type of *IGCDCT. A system-supplied and a user-created dictionary are used with DBCS conversion.

System-Supplied Dictionary (for Japanese Use Only)

QSYSIGCDCT, the system-supplied dictionary that is stored in the library, QSYS, is a collection of entries with a Japanese pronunciation expressed in alphanumeric

characters and the DBCS words related to those entries. The system checks this dictionary second when performing DBCS conversion.

QSYSIGCDCT contains these entries:

- Personal names
  - Family names
  - First names
- Organization names
  - Private enterprises registered in the security market
  - Public corporations
  - Typical organizations in the central and
  - local governments
  - Most universities and colleges
- Addresses
  - Public administration units within the prefectures
  - Towns and streets in 11 major cities
- Business terms, such as department names and position titles commonly used in enterprises
- Individual double-byte characters, including basic double-byte characters, as defined by IBM

You cannot add or delete entries from this dictionary. However, you may rearrange the related DBCS words so that the words used most frequently are displayed first during DBCS conversion. See “Editing a DBCS Conversion Dictionary” on page 208 for instructions on rearranging terms.

User-Created Dictionary

A user-created dictionary contains any alphanumeric entries and related DBCS words that you choose to include. You might create a user dictionary to contain words unique to your business or words that you use regularly but that are not included in the system-supplied dictionary.

You can create one or more DBCS conversion dictionaries with any name and store them in any library. When performing DBCS conversion, however, the system only refers to the first user dictionary named QUSRIGCDCT in the user’s library list, no matter how many dictionaries you have or what they are named. Make sure that the library list is properly specified so that the system checks the correct dictionary.

During DBCS conversion, the system checks QUSRIGCDCT before checking QSYSIGCDCT.

Commands for DBCS Conversion Dictionaries

You can use the following commands to perform object management functions with the DBCS conversion dictionary. Specify the OBJTYPE(*IGCDCT) parameter when entering these commands:

- CHGOBJOWN: Change the owner of a DBCS conversion dictionary
- CHKOBJ: Check a DBCS conversion dictionary
- CRTDUPOBJ: Create a duplicate object of the dictionary
- DMPOBJ: Dump a DBCS conversion dictionary

- DMPSYSOBJ: Dump the system-supplied dictionary
- DSPOBJAUT: Display a user's authority to the dictionary
- GRTOBJAUT: Grant authority to use the dictionary
- MOVOBJ: Move the dictionary to another library
- RNMOBJ: Rename the dictionary
- RSTOBJ: Restore the dictionary
- RVKOBJAUT: Revoke authority to use the dictionary
- SAVOBJ: Save the dictionary
- SAVCHGOBJ: Save a changed dictionary

The system saves or restores DBCS conversion dictionaries when you use these commands:

- RSTLIB: Restore a library in which the dictionary is stored
- SAVLIB: Save a library in which the dictionary is stored
- SAVSYS: Save QSYSIGCDCT, the system DBCS conversion dictionary, when saving the system

You can use the following commands to create, edit, display, and delete a dictionary:

- CRTIGCDCT: Create DBCS Conversion Dictionary
- EDTIGCDCT: Edit DBCS Conversion Dictionary
- DSPIGCDCT: Display DBCS Conversion Dictionary
- DLTIGCDCT: Delete DBCS Conversion Dictionary

Creating a DBCS Conversion Dictionary

To create a DBCS conversion dictionary, do the following:

1. Use the Create DBCS Conversion Dictionary (CRTIGCDCT) command.
2. Name the dictionary, QUSRIGCDCT, so it can be used during DBCS conversion. The system uses the dictionary if it is the first user-created dictionary found when searching a user's library list.

   You might call the dictionary by another name while it is being created to prevent application programs from using it for conversion. Later, change the dictionary name using the Rename Object (RNMOBJ) command.

   For example, to create a user DBCS conversion dictionary to be stored in the library DBCSLIB, enter:

   ```
   CRTIGCDCT IGCDCT(DBCSLIB/QUSRIGCDCT)
   ```
3. Use the EDTIGCDCT command to put entries and related words into the dictionary after creating it. See "Editing a DBCS Conversion Dictionary" for instructions on putting entries in the dictionary.

Editing a DBCS Conversion Dictionary

Use the Edit DBCS Conversion Dictionary (EDTIGCDCT) command to edit the DBCS conversion dictionary. Use editing to add user-defined characters to the dictionary, so that users can enter characters using DBCS conversion, and rearrange terms in a DBCS conversion dictionary to suit individual needs.

Requirements for a DBCS Conversion Dictionary: The display station needed for use while editing the DBCS conversion dictionary depends on the value that you entered for the ENTRY parameter on the EDTIGCDCT command:

- If you specified a specific string with the ENTRY parameter or if you want to display double-byte characters, you must use a DBCS display station.
- If you did not specify a specific string with the ENTRY parameter, or if you do not want to display double-byte characters, use either a DBCS display station, or a 24-row by 80-column alphanumeric display station.

DBCS Conversion Dictionary Operations: You may perform the following editing operations on a user-created DBCS conversion dictionary:

- Add entries to the dictionary (including adding the first entries to the dictionary after it is created). The dictionary can contain as many as 99,999 entries.
- Delete entries from the dictionary.
- Change entries in the dictionary, such as replacing the DBCS words related to an alphanumeric entry.
- Move the DBCS words related to an alphanumeric entry to rearrange the order in which they appear during DBCS conversion.

The only editing function that you can perform with QSYSIGCDCT, the system-supplied dictionary, is to move DBCS words related to an alphanumeric entry. Move words in order to rearrange the order in which they appear during DBCS conversion.

Displays Used for Editing a DBCS Conversion Dictionary: After you enter the EDTIGCDCT command, the system presents either the Work With DBCS Conversion Dictionary display or the Edit Related Words display, depending on the value entered for the ENTRY parameter on the command.

Work with DBCS Conversion Dictionary Display: Use the display in Figure 19 on page 210 to work with alphanumeric entries, such as choosing an entry to edit or deleting an entry. The system displays the Work with DBCS Conversion Dictionary display if you enter *ALL or a generic string for the ENTRY parameter of the EDTIGCDCT command.

See the discussion of the EDTIGCDCT command in the *CL Reference (Abridged)* for a complete description of the Work with DBCS Conversion Dictionary display.

```
                        漢字変換辞書の処理

 辞書 . . . . . :   QSYSIGCDCT      項目 . . . . :   *ALL
   ライブラリー . . :     QSYS

 サブセット  .   __________   *ALL, 総称*

 オプション（および項目）を打鍵して実行キーを押してください。
  1= 追加   2= 編集   4= 削除   5= 表示   6= 印刷

OPT  項目          OPT  項目          OPT  項目          OPT  項目
 _   __________
 _   ?             _   ｱｯｻﾌﾞﾁｮｳ       _   ｱｯﾊﾞｸ          _   ｱｲｵｲﾁｮｳ
 _   ｱ             _   ｱｯｼｭｸ         _   ｱｱ            _   ｱｲｵｲﾄﾞｵﾘ
 _   ｱｯ            _   ｱｯｼｭｸｷﾁｮｳ    _   ｱｱｲｳ         _   ｱｲｵﾁｮｳ
 _   ｱｯｶ           _   ｱｯｼｭｸﾋ        _   ｱｲ            _   ｱｲｶﾜ
 _   ｱｯｹﾝｸﾞﾝ       _   ｱｯｼｮｳ         _   ｱｲｲﾄﾅﾑ       _   ｱｲｶﾜﾁｮｳ
 _   ｱｯｹﾝﾁｮｳ      _   ｱｯｽﾙ          _   ｱｲｳﾁ         _   ｱｲｶﾝ
 _   ｱｯｻｲ          _   ｱｯｾｲ         _   ｱｲｳﾗ         _   ｱｲｶﾞ
 _   ｱｯｻｸ          _   ｱｯｾﾝ         _   ｱｲｵｲ         _   ｱｲｶﾞﾝ
 _   ｱｯｻﾂ          _   ｱｯﾄｳ         _   ｱｲｵｲｼ        _   ｱｲｷ
                                                          続く ...
F3= 終了    F5= 再表示    F12= 取消し
```

HRSLS332-1

Figure 19. Display for Work with DBCS Conversion Dictionary

Edit Related Words Display: Use this display to work with the DBCS words related to an alphanumeric entry. The system displays the Edit Related Words display if you enter a specific string for the ENTRY parameter. The system also displays the Edit Related Words display if you choose an entry to edit from the Work with DBCS Conversion Dictionary display. Figure 20 on page 211 is an example of the Edit Related Words display.

See the discussion of the EDTIGCDCT command in the *CL Reference (Abridged)* for a complete description of the Edit Related Words display.

Examples of Editing Operations: The following sections give examples of the editing operations that you can perform using the EDTIGCDCT displays:

- Beginning to edit a dictionary
- Adding the first entries in a dictionary
- Deleting an entry
- Moving a related word
- Ending editing the dictionary

Beginning to Edit a Dictionary: Enter the EDTIGCDCT command to start editing the dictionary for any type of editing operation. For example, to put the first entry in the dictionary, enter:

```
EDTIGCDCT IGCDCT(DBCSLIB/QUSRIGCDCT) +
  ENTRY(*ALL)
```

Or, to edit the entries beginning with the string ABC enter:

```
EDTIGCDCT IGCDCT(DBCSLIB/QUSRIGCDCT) +
  ENTRY('ABC*')
```

Adding the First Entries in a Dictionary: To add the first entries into a dictionary, do the following:

1. Specify ENTRY(*ALL) when entering the EDTIGCDCT command. For example, to edit the dictionary QUSRIGCDCT stored in the library DBCSLIB, enter:

   ```
   EDTIGCDCT IGCDCT(DBCSLIB/QUSRIGCDCT) +
     ENTRY(*ALL)
   ```

   The system displays the Work with DBCS Conversion Dictionary display.
2. Enter a 1 in the first option field in the list and enter an alphanumeric entry to be added to the dictionary in the entry field.

   The system then displays the Edit Related Words display showing only two lines of data: BEGINNING OF DATA and END OF DATA.
3. Enter an I in the *NBR* field beside the BEGINNING OF DATA line to insert a line.
4. Press the Enter key. The system displays a blank line.

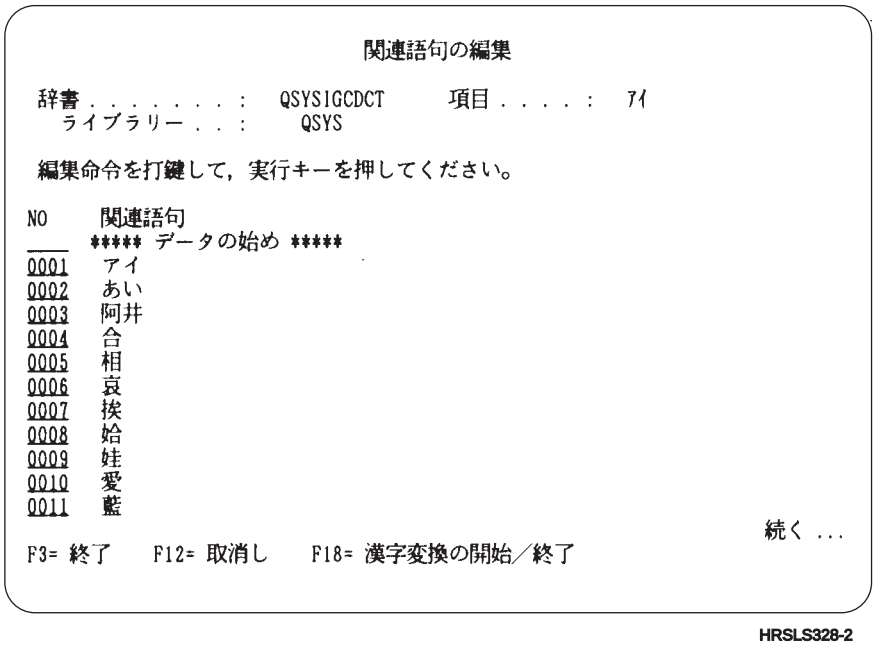


Figure 20. Display for Edit Related Words

5. On the blank line, enter a DBCS word to be related to the new alphanumeric entry.

   If you enter data on the inserted line and leave the cursor on that line, another new line appears below when you press the Enter key. You can enter another DBCS word on this line, or delete it by leaving it blank, and pressing the Enter key.
6. When you finish adding this first entry, press F12 to get the Exit Dictionary Entry display. Enter the Y option to save the entry and then return to the Work With DBCS Conversion Dictionary display. Enter option 1 again and enter another alphanumeric entry in the entry field to continue adding entries to the dictionary, or press F3 to end editing the dictionary.

Moving a Related Word: Moving the words related to an alphanumeric entry changes the order in which the words appear during DBCS conversion. To move a word, do the following:

1. Display the Edit Related Words display for the entry in which you want to move DBCS words, either by entering a specific entry with the EDTIGCDCT command, or by choosing an entry to edit from the Work with DBCS Conversion Dictionary display.
2. When the display appears, enter an M in the *NBR* field beside the DBCS word to be moved.
3. Enter an A in the *NBR* field of the line after which the word will be moved.
4. Press the Enter key. The system moves the word on the line marked M to a position immediately following the line marked with an A.

Deleting an Entry: Enter a 4 in the input field beside the entry to be deleted as shown in Figure 21.

Ending the Editing Process: To end the editing operation, press F3. The Exit Dictionary Entry display is displayed, and you can choose to save the entry or not. The system then returns you to your basic working display, such as the Command Entry display.

Suggestions for Editing the Dictionary: When editing the DBCS conversion dictionary, consider the following:

- You can use DBCS conversion with the Edit Related Words display to enter related words into a user-created dictionary. See “DBCS Conversion (for Japanese Use Only)” on page 214 for information on this procedure.

```
                          漢字変換辞書の処理

 辞書 . . . . . :   QSYSIGCDCT      項目 . . . . :   *ALL
   ライブラリー . . :   QSYS

 サブセット  .   _____________   *ALL, 総称*

 オプション（および項目）を打鍵して実行キーを押してください。
  1= 追加   2= 編集   4= 削除   5= 表示   6= 印刷

OPT  項目          OPT  項目          OPT  項目          OPT  項目
 _   _____________
 _   ?              _   ｱｯｻﾌﾞﾁｮｳ      _   ｱｯﾊﾟｸ        _   ｱｲｵｲﾁｮｳ
 _   ｱ              _   ｱｯｼｭｸ        _   ｱｱ          _   ｱｲｵｲﾄﾞｵﾘ
 _   ｱｯ             _   ｱｯｼｭｸｷﾁｮｳ    _   ｱｱｲｳ       _   ｱｲｵﾁｮｳ
 _   ｱｯｶ            _   ｱｯｼｭｸﾋ       _   ｱｲ          _   ｱｲｶﾜ
 _   ｱｯｹﾝｸﾞﾝ        _   ｱｯｼｮｳ        _   ｱｲｲﾄﾅﾑ      _   ｱｲｶﾜﾁｮｳ
 _   ｱｯｹﾝﾁｮｳ       _   ｱｯｽﾙ         _   ｱｲｳﾁ        _   ｱｲｶﾝ
 _   ｱｯｻｲ          _   ｱｯｾｲ         _   ｱｲｳﾗ        _   ｱｲｶﾞ
 _   ｱｯｻｸ          _   ｱｯｾﾝ         _   ｱｲｵｲ        _   ｱｲｶﾞﾝ
 _   ｱｯｻﾂ          4   ｱｯﾄｳ         _   ｱｲｵｲｼ       _   ｱｲｷ
                                                          続く ...
F3= 終了    F5= 再表示    F12= 取消し
```

HRSLS331-2

Figure 21. Display for Deleting a Conversion Dictionary Entry

- Place the most commonly used DBCS words at the beginning of the list of related words on the Edit Related Words display. This simplifies DBCS conversion because the system displays the related words in the same order in which those words are listed in the dictionary.

Displaying and Printing the DBCS Conversion Dictionary

Use the Display DBCS Conversion Dictionary (DSPIGCDCT) command to display and print the DBCS conversion dictionary. You can display or print the entire dictionary or just a certain part of it, depending on the value you specify for the ENTRY parameter.

For example, to print the entry ABC from the dictionary QUSRIGCDCT and its related words, enter:

```
DSPIGCDCT IGCDCT(DBCSLIB/QUSRIGCDCT)  +
  ENTRY(ABC) OUTPUT(*PRINT)
```

To display all of the entries from the system-supplied dictionary QSYSIGCDCT and their related words, enter:

```
DSPIGCDCT IGCDCT(QSYS/QSYSIGCDCT)
```

Figure 22 on page 214 provides an example of the display produced by the DSPIGCDCT command. It shows alphanumeric entries and their related words.

See the discussion of the DSPIGCDCT command in the *CL Reference (Abridged)* for a complete description of the command and the display it produces.

Deleting a DBCS Conversion Dictionary

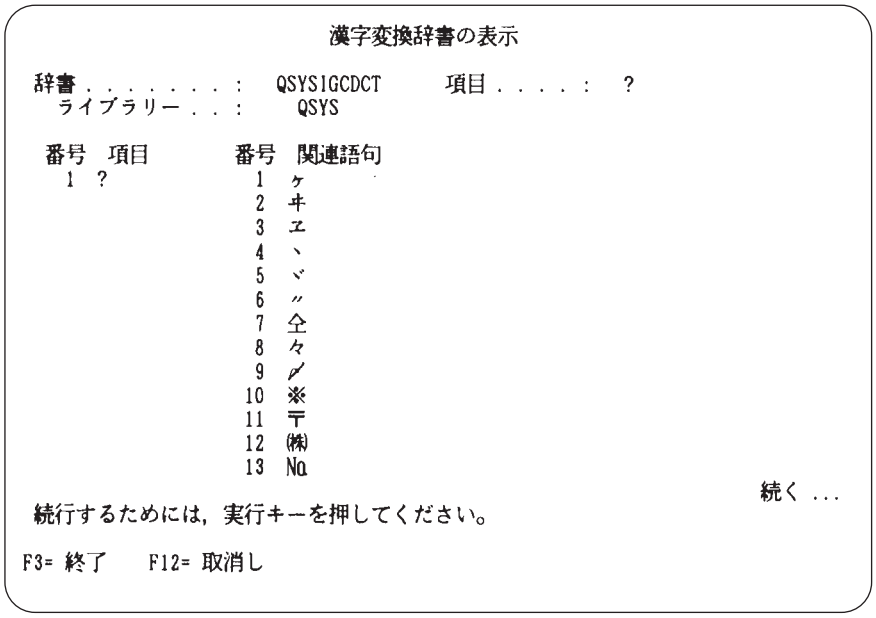
Use the Delete DBCS Conversion Dictionary (DLTIGCDCT) command to delete a DBCS conversion dictionary from the system.

In order to delete the dictionary, you must have object existence authority to the dictionary and object operational authorities to the library in which the dictionary is stored.

When you delete a dictionary, make sure that you specify the correct library name. It is possible that many users have their own dictionaries, each named QUSRIGCDCT, stored in their libraries. If you do not specify any library name, the system deletes the first DBCS conversion dictionary in your library list.

For example, to delete the DBCS conversion dictionary QUSRIGCDCT in the library DBCSLIB, enter:

```
DLTIGCDCT IGCDCT(DBCSLIB/QUSRIGCDCT)
```



HRSLS337-2

Figure 22. Display Produced by the DSPIGCDCT Command

DBCS Conversion (for Japanese Use Only)

When you use DBCS display stations to enter double-byte data, you may use the various data entry methods supported on the display station, or you may choose to use the AS/400 DBCS conversion support. DBCS conversion lets you enter an alphanumeric entry or DBCS code and convert the entry or code to its related DBCS word. DBCS conversion is intended for Japanese character sets and its use is limited for application to other double-byte character sets.

Specifically, DBCS conversion lets you convert the following:

- A string of alphanumeric characters to a DBCS word
- English alphanumeric characters to double-byte alphanumeric characters
- Alphanumeric Katakana to double-byte Hiragana and Katakana letters
- A *DBCS code* to its corresponding double-byte character
- A *DBCS number* to its corresponding double-byte character

Where You Can Use DBCS Conversion

You can use DBCS conversion in the following instances:

- When entering data into input fields of certain SEU displays. For information about which fields you can use with DBCS conversion, refer to the *ADTS for AS/400: Source Entry Utility* book.
- When prompting for double-byte data using QCMDEXEC. For instructions on this procedure, see the *CL Reference (Abridged)*.
- When entering data into input fields of DBCS display files in user-written applications. Specify DBCS conversion with the DDS keyword IGCCNV. See the *DDS Reference* for information on this keyword.

- When editing the related words on the Edit Related Words display, which is displayed when editing the DBCS conversion dictionary (EDTIGCDCT command). See “Editing a DBCS Conversion Dictionary” on page 208 for information on the Edit Related Words display.

How DBCS Conversion Works

DBCS conversion is an interactive function between you and the system in which you enter an alphanumeric entry. The system displays related DBCS words, and you choose which word to use.

The system determines which words are related to an alphanumeric entry by checking DBCS conversion dictionaries. The system checks two DBCS conversion dictionaries when performing DBCS conversion. It checks the first user-created dictionary named QUSRIGCDCT found when searching a user's library list. Then, it checks the system-supplied dictionary, QSYSIGCDCT, stored in the library QSYS. (QSYSIGCDCT contains only Japanese double-byte characters.) You can create other user dictionaries, and you can give them names other than QUSRIGCDCT, but the system only refers to the first user-created dictionary named QUSRIGCDCT found in your library list when performing DBCS conversion.

After checking the dictionaries, the system displays words related to the alphanumeric entry. You then position the cursor under the word of your choice, and press the Enter key. The system enters that word where the cursor was positioned when you began DBCS conversion.

Using DBCS Conversion

You can change the user-defined dictionary used during DBCS conversion. Before you change the user-defined dictionary, end your application program or end the command that the system is performing. Then change the dictionary that is used by changing the library list (using the CHGLIBL command).

You can create your own DBCS conversion dictionary for DBCS conversion. The system-supplied dictionary is a collection of entries with a Japanese pronunciation expressed in alphanumeric characters and Japanese DBCS words related to the entry. See “Creating a DBCS Conversion Dictionary” on page 208 for instructions on this procedure.

If no user-created dictionary is found, the system refers only to QSYSIGCDCT. See “DBCS Conversion Dictionaries” on page 206 for more information on creating and using DBCS conversion dictionaries.

Performing DBCS Conversion

The following procedure describes how to convert one alphanumeric entry to its related DBCS word using DBCS conversion. You must start DBCS conversion separately for each field in which you want to enter double-byte data.

Note: DBCS conversion is intended for Japanese data entry. Its use with other languages is limited.

While performing DBCS conversion, you can display information about the function by pressing the Help key. Help is available until you end DBCS conversion.

1. Position the cursor in the field in which you want to enter double-byte characters. Insert shift-control characters into the field if they have not yet been inserted. To find out how to insert shift characters, see “Inserting Shift-Control Characters” on page 194.
2. Position the cursor under the shift-in character, in a blank area between the shift-control characters, or under a double-byte character.
3. Press the function key used to start DBCS conversion.

   In SEU, as well as from the Edit Related Words display (displayed when using the EDTIGCDCT command), press F18. The system displays the following prompt line:

   ```
   A _____ C
     B
   ```

4. Enter the following values:
   a. In the field marked A, enter one of the following:

      I Inserts the converted word before the character under which you positioned the cursor in step 2.

      R Replaces the character under which you positioned the cursor in step 2 with the converted word.

   b. In the field marked B, enter one of the following:
      1) A string of alphanumeric characters to be converted. The string can have as many as 12 characters.
      2) The 4-character DBCS code of a double-byte character.
      3) The 2- to 5-digit DBCS number of a double-byte character.

   c. In the field marked C enter one of the following conversion codes:

      No entry
      Converts the entry in field B from alphanumeric to double-byte by referring to the DBCS conversion dictionaries.

      G Converts the 2- to 5-digit DBCS number in field B to the character it represents.

      H Converts the entry in field B to double-byte Hiragana, uppercase alphabetic, numeric, or special characters.

      K Converts the entry in field B to double-byte Hiragana, lowercase alphabetic, numeric, or special characters.

      X Converts the 4-character DBCS code to the character it represents.
5. Press the Enter key. The system displays the following prompt line:

   ```
   A _____ C _____+
     B         D
   ```

6. In the field marked D, the system displays words related to the entry in field B.

   If you see a plus (+) sign following the last displayed word, the system has additional words to display. Press the Roll Up key to see these entries. Then, to return to a word displayed earlier, press the Roll Down key.

   If a word is shown in a reverse image, the word contains an embedded blank.
7. Choose the DBCS word from field D that best suits your needs by positioning the cursor under that DBCS word.
8. Press the Enter key. The system enters the word where the cursor was positioned in step 2, either by inserting the word or by replacing another word, depending on what you entered in field A.
9. Do one of the following:

a. Continue using DBCS conversion. Repeat 4 on page 216 through 8 on page 216 until you finish entering data into the field.

b. End DBCS conversion by pressing the *same* function key used to start conversion. The system automatically ends conversion when you reach the end of the field.

   In SEU, as well as from the Edit Related Words display (displayed when using the EDTIGCDCT command), press F18.

   Note: Until DBCS conversion is ended, you cannot perform any other system function. For example, the F3 key cannot be used to exit an SEU display.

Examples

Converting One AIphanumeric Entry to a Double-Byte Entry: The following example shows how to convert one entry and enter it into a field.

1. Position the cursor in the field in which you want to enter double-byte data (see Figure 23 on page 218).
2. Insert shift-control characters into the field. See “Inserting Shift-Control Characters” on page 194 for instructions on inserting shift-control characters.
3. Press the function key used to start DBCS conversion. For the display just shown, the function key is F18. The system displays a prompt line as shown in Figure 24 on page 218.

   Because the cursor was placed under a shift-in character when conversion was started, conversion automatically is set to I (inserting the converted word).
4. Enter an alphanumeric entry to be converted in the second field.

   Leave the third field blank. See the example screen in Figure 25 on page 219.
5. Press the Enter key. The system displays related DBCS words.
6. Position the cursor under the DBCS word that you want to enter, if that word is not the first DBCS word shown. In the example screen shown in Figure 26 on page 219, the first word is the one to be entered.
7. Press the Enter key. The DBCS word is entered into the field as shown in Figure 27 on page 220.

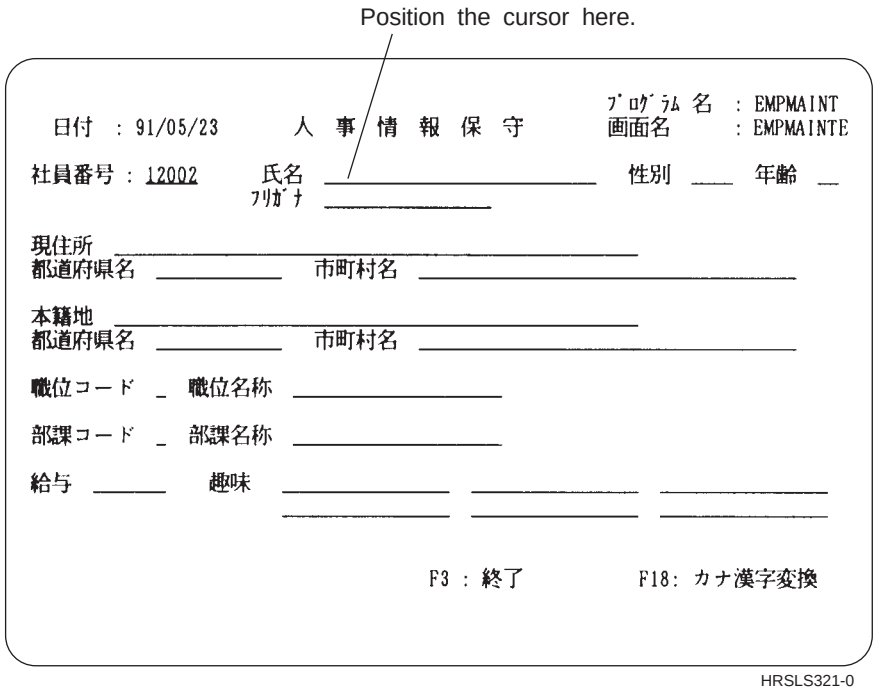


Figure 23. Example Screen 1

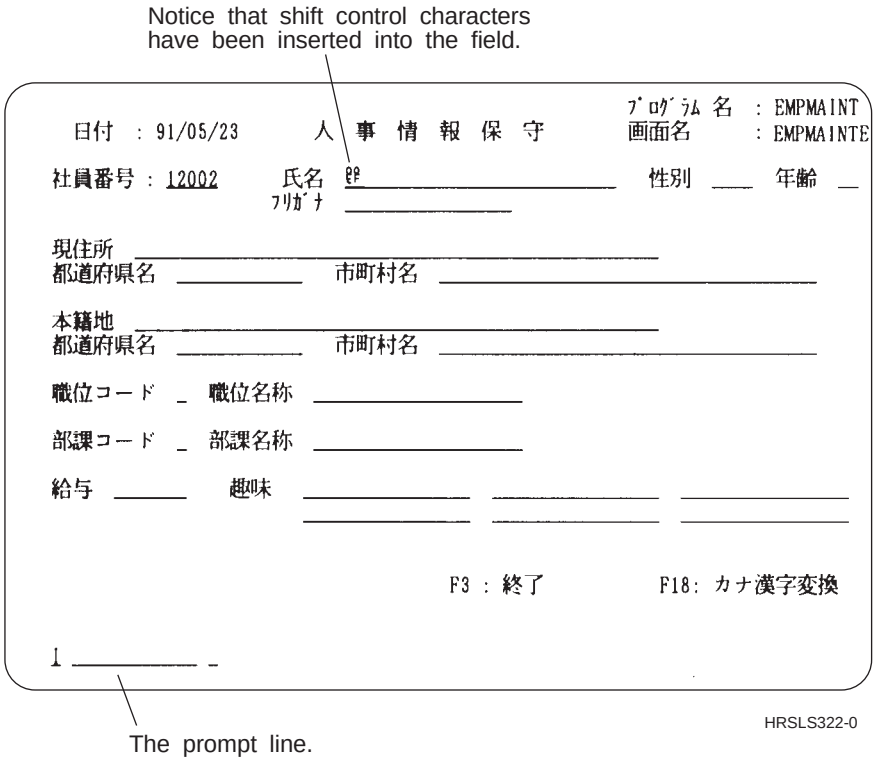


Figure 24. Example Screen 2

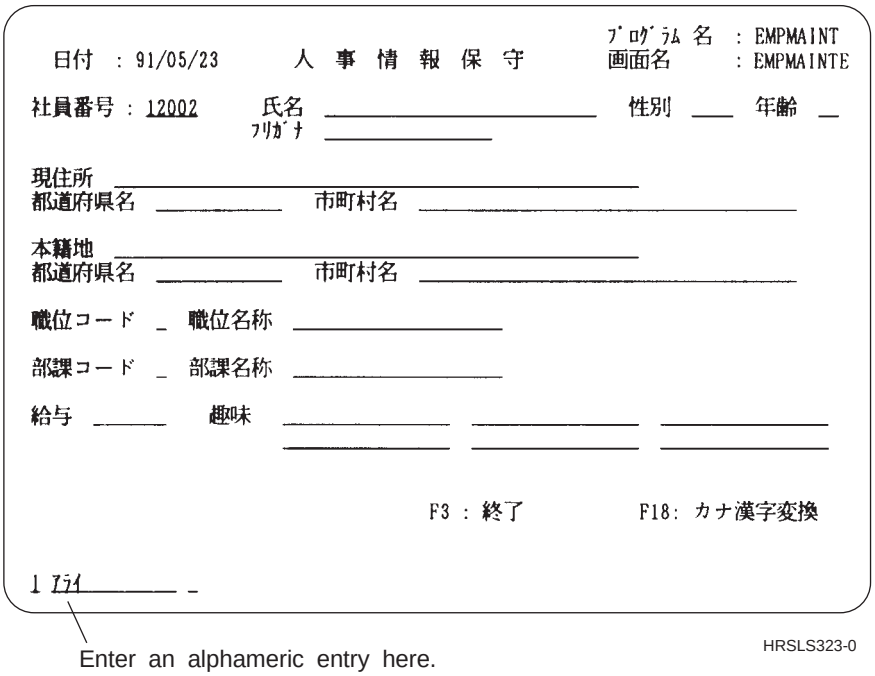


Figure 25. Example Screen 3

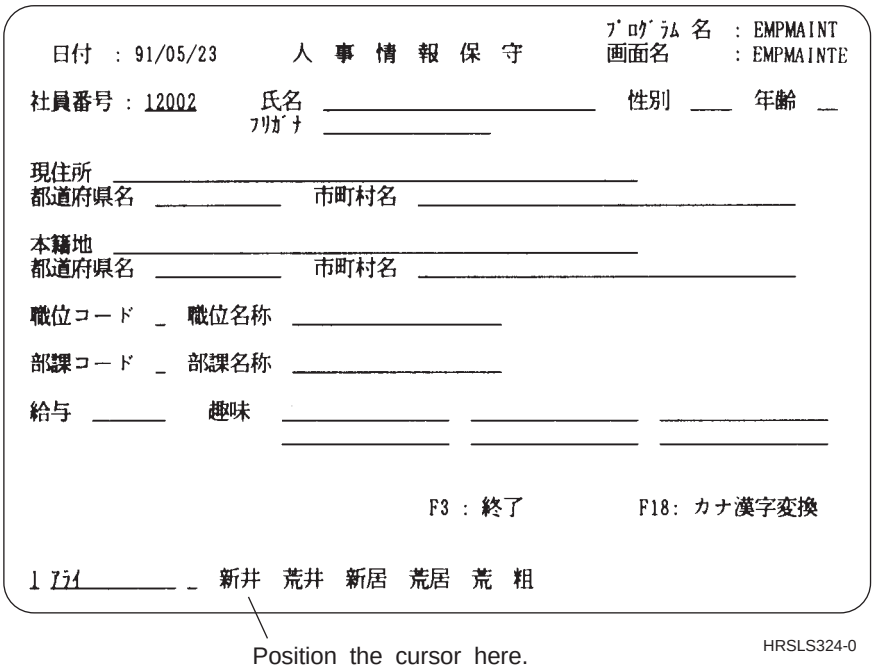
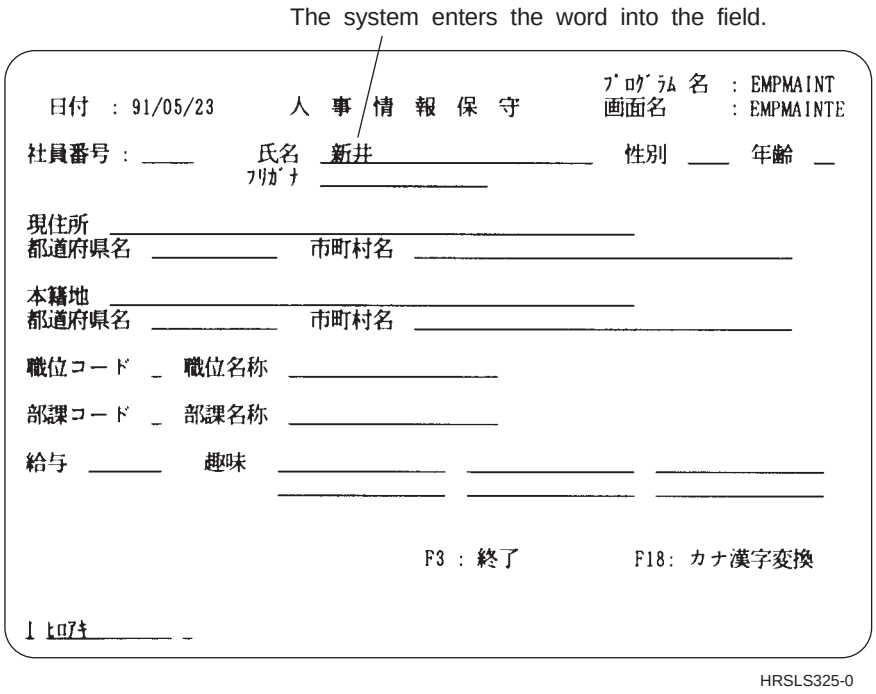


Figure 26. Example Screen 4



HRSLS325-0

Figure 27. Example Screen 5

Converting Many Alphanumeric Entries at One Time: You do not have to continually start DBCS conversion for each alphanumeric entry. Instead, you can do the following:

1. Enter as many alphanumeric entries as will fit into field B. Separate each entry by a blank. Field B contains space for 12 alphanumeric characters:

```
 These are the entries to be converted.
    |   |   |
  I XXX_YYY_ZZZ_ _
  A      B       C                  D
```

   The system converts the entries one at a time, in the order entered. When the system converts an entry, the system displays the DBCS words related to that entry in field D.
2. Position the cursor under the DBCS word that you want to use.
3. Press the Enter key. Then, the system adjusts field B; the next entry is moved to the position farthest left of the field. The DBCS words related to that entry are displayed in field D.

   At this time, you can enter additional alphanumeric entries to be converted at the end of field B.

Converting Alphanumeric Blanks to DBCS Blanks: You can convert alphanumeric blanks (one position wide) to DBCS blanks (two positions wide, the same width as double-byte characters) using DBCS conversion.

To convert blanks, do the following:

1. Enter one or more blanks in field B.

```
_ ___________ _
A      B      C              D
```

2. Press the Enter key. The system displays in field D the same number of DBCS blanks as the alphanumeric blanks that you entered in field B. The DBCS blanks are displayed in reverse image.
3. Press the Enter key again. The system enters the DBCS blanks into the field where you started DBCS conversion.

Changing Alphanumeric Entries or Conversion Code: If none of the related words shown during conversion are suitable candidates for the alphanumeric entry, and you would like to try a conversion again (by using a different type of conversion or a different alphanumeric entry), do the following:

1. Move the cursor to field B. For example:

```
Move the cursor here.
  |
  XXXXXX

_ __________ _ ____________________
A      B       C          D
```

2. Do one of the following:
   a. Position the cursor under the first position of the field in which you want to enter alphanumeric entries.
   b. Enter a different alphanumeric entry.
   c. Change the conversion code in field C, such as from H to K.
3. Press the Enter key.
4. Continue DBCS conversion.

Using DBCS Conversion to Enter Words in the DBCS Conversion Dictionary: You can use DBCS conversion when entering DBCS words on the Edit Related Words display.

To start DBCS conversion, do the following:

1. Position the cursor at the position where the DBCS word is to be entered.
2. Press F18. The system displays the conversion prompt line at the bottom of the display.

Perform DBCS conversion according to the instructions described in “Performing DBCS Conversion” on page 215.

Note: You must start and end DBCS conversion separately for each line of data.

Considerations for Using DBCS Conversion

Consider the following when performing DBCS conversion:

- You can only perform DBCS conversion at a DBCS display station, using the 5556 keyboard.
- You may use DBCS conversion to insert or replace characters only if the line in which double-byte characters are to be inserted has sufficient space.
  – The space available for inserting characters is equal to the number of characters from the last character on the line that is not blank to the right edge of the display.
  – The space available for replacing characters is equal to the number of characters from the cursor position (including the character marked by the cursor) to the end of the DBCS portion of the field.

  The following happens when you do not have enough space:

- If you try to insert or replace a string of characters where there is no space available, the system sends a message.
- If you ignore the message and press the Enter key again, the system truncates the characters in excess of the limit from the right side of the string to be inserted or replaced.

Bibliography

The following AS/400 books contain information you may need. The books are listed with their full title and base order number.

Planning, Installation, and Migration

- *National Language Support*, SC41-5101-01, provides the data processing manager, system operator and manager, application programmer, end user, IBM marketing representative, and system engineer with informaiton required to understand and use the national language support function on the AS/400 system. This book prepares the AS/400 user for planning, installing, configuring, and using AS/400 national language support (NLS) and multilingual support of the AS/400 system. It also provides an explanation of the database management of multilingual data and application considerations for a multilingual system.
- *Local Device Configuration*, SC41-5121-00, provides the system operator or system administrator with information on how to do an initial local hardware configuration and how to change that configuration. It also contains conceptual information for device configuration, and planning information for device configuration on the 9406, 9404, and 9402 System Units.

Application Development

- *ADTS/400: Character Generator Utility*, SC09-1769-00, provides the application programmer or system programmer with information about using the Application Development Tools character generator utility (CGU) to create and maintain a double-byte character set (DBCS) on the system.
- *ADTS for AS/400: Screen Design Aid*, SC09-2604-00, provides the application programmer, system programmer, or data processing manager with information about using the Application Development Tools screen design aid (SDA) to design, create, and maintain displays, menus, and online help information.
- *ADTS for AS/400: Source Entry Utility*, SC09-2605-00, provides the application programmer or system programmer with information about using the Application Development Tools source entry utility (SEU) to create and edit source members.
- *System/36-Compatible COBOL User's Guide and Reference*, SC09-1815-00, provides information about using COBOL in the System/36 environment on the AS/400 system. It provides information on how to program in COBOL in the AS/400 System/36 environment and how to use existing System/36 COBOL programs.
- *System/36-Compatible RPG II User's Guide and Reference*, SC09-1818-00, provides programming information for the RPG II language on the AS/400 system for those who have a basic understanding of data processing concepts and of the RPG II language. It explains how to design, code, enter, compile, test, and run RPG II programs.

System Management

- *Backup and Recovery*, SC41-5304-03, provides the system programmer with information to plan a backup and recovery strategy. Also included are procedures to implement your backup and recovery strategy, how to recover from disk unit failures, and how to recover from a site loss.
- *Backup and Recovery*, SC41-5304-03, provides the system programmer with information about starting checksum or mirrored protection. It also tells how to add disk units to an existing auxiliary storage pool (ASP) and describes device parity protection.
- *Work Management*, SC41-5306-03, provides information about how to create and change a work management environment.
- *Security - Reference*, SC41-5302-03, provides the system programmer with information about planning, designing, and auditing security. Includes information about security system values, user profiles, and resource security.
- *Security - Basic*, SC41-5301-00, provides basic information about planning and setting up security on the AS/400 system.

Communications and Connectivity

- *ICF Programming*, SC41-5442-00, provides the application programmer with the information needed to write application programs that use AS/400 communications and ICF files. It also contains information on data description specifications (DDS) keywords, system-supplied formats, return codes, file transfer support, and programming examples.

Program Enablers

- *DDS Reference*, SC41-5712-01, provides the application programmer with detailed descriptions of the entries and keywords needed to describe database files (both logical and physical) and certain device files (for displays, printers, and ICF) external to the user's programs.
- *DB2 UDB for AS/400 Database Programming*, SC41-5701-02, provides the application programmer or system programmer with a detailed discussion of the AS/400 database organization, including information on how to create, describe, and manipulate database files on the system.
- *Application Display Programming*, SC41-5715-00, provides information about creating and maintaining screens for applications, creating online help information, and working with display files on the AS/400 system.
- *Printer Device Programming*, SC41-5713-03, provides information on how to understand and control printing: printing elements and concepts, printer file support, print spooling support, printer connectivity, advanced function printing, and printing with personal computers.
- *Tape and Diskette Device Programming*, SC41-5716-01, provides information about creating and maintaining tape device files and diskette device files.
- *System/36 Environment Programming*, SC41-4730-00, provides information identifying the functional and operational differences between the applications process on the System/36 and in the System/36 environment on the AS/400 system.
- *CL Programming*, SC41-5721-02, provides a wide-ranging discussion of programming topics, including a general discussion of objects and libraries, control language (CL) programming, controlling flow and communicating between programs, working with objects in CL programs, and creating CL programs. Other topics include predefined and immediate messages and message handling, defining and creating user-defined commands and menus, and application testing, including debug mode, breakpoints, traces, and display functions.
- *CL Reference (Abridged)*, SC41-5722-03, provides a description of the control language (CL) and its commands. Each command is defined including its syntax diagram, parameters, default values, and keywords.

System Management

- *Distributed Data Management*, SC41-5307-00, provides the application programmer or system programmer with information about remote file processing. It describes how to define a remote file to OS/400 distributed data management (DDM), how to create a DDM file, which file utilities are supported through DDM, and the requirements of OS/400 DDM as related to other systems.

System Use

- *System Operations: Font Management Aid User's Guide,* SC18-2216, provides information about how to maintain or use double-byte character set (DBCS) user-defined characters. It also shows how to maintain and distribute user-defined characters and associated data entry dictionaries between the AS/400 system and its DBCS display stations.
- *System Operation*, SC41-4203-00, is intended for system operators or administrators and includes information on operating the console, starting and stopping the system, responding to messages, and controlling jobs and printer output. It also provides information on devices, communications, tapes and diskettes, using online education and electronic customer support, managing system operations, working with program temporary fixes and handling problems.
- *Basic System Operation, Administration, and Problem Handling*, SC41-5206-03, is intended for system operators or administrators and includes information about the system unit control panel, starting and stopping the system, using tapes and diskettes, working with program temporary fixes, as well as handling problems.

- *Using OfficeVision/400 Word Processing*, SH21-0701-00, provides the office user with information on how to use the word processing functions of SAA OfficeVision, and it can be used with the OfficeVision online information.
- *IDDU Use*, SC41-5704-00, provides the administrative secretary, business professional, or programmer with detailed information on how to use OS/400 interactive data definition utility (IDDU) to describe data dictionaries, files, and records to the system.
- *Utilities: Kanji Print Function User's Guide and Reference* SH18-2179, provides information about using the Kanji printer function (KPF) to create and maintain symbols and tailored forms.

Tutorial

- *Experience RPG IV Workbook, SC09-1938*, is an interactive self-study program explaining the differences between RPG III and RPG IV and how to work within the new ILE environment. An accompanying workbook provides additional exercises and doubles as a reference upon completion of the tutorial. ILE RPG/400 code examples are shipped with the tutorial and run directly on the AS/400. Dial 1-800-IBM-CALL to order the tutorial.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Readers' Comments — We'd Like to Hear from You

AS/400e
Data Management
Version 4

Publication No. RBAL-3000-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

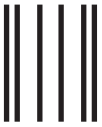
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
RBAL-3000-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM CORPORATION
ATTN DEPT 542 IDCLERK
3605 HWY 52 N
ROCHESTER MN 55901-7829

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.

RBAL-3000-00