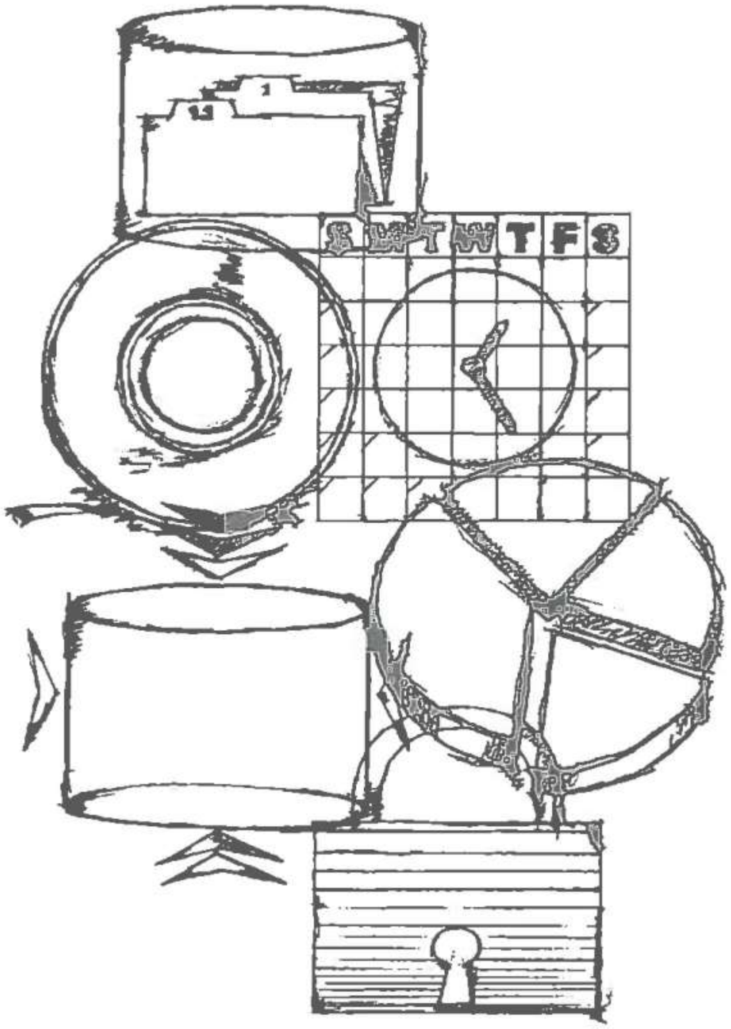
LIBRARIAN/iX™

Administrator's Guide

Version 4.00
May 1998



OCS

Quality • Innovation • Service

LIBRARIAN/iX Administrator's Guide
Version 4.00

Table of Contents

List of Figures

List of Tables

Preface

Purpose of This Manual

The *LIBRARIAN/iX Administrator's Guide* describes how to setup and maintain LIBRARIAN. It is the companion piece to the *LIBRARIAN/iX Reference Guide* and the *LIBRARIAN/iX User's Guide*.

Audience

This manual is written for personnel responsible for setting up and maintaining the LIBRARIAN database of rules. Knowledge of basic operating system concepts and terminology is assumed. No previous knowledge of LIBRARIAN is required.

How This Manual is Organized

The *LIBRARIAN/iX Administrator's Guide* chapters are organized as follows:

Preface	The chapter you are now reading: how to use this guide.
Chapter 1	"Introduction": an overview of typical file management policies and objectives and how to configure LIBRARIAN.
Chapter 2	"Getting Started with Basic Rules": applying the Shortcut program to get started using LIBRARIAN.
Chapter 3	"Master Library": master library concepts and how to define a library in the database.
Chapter 4	"File Movement Rules": how to define rules to determine how users obtain copies, replace master files with revisions, and perform other file operations.
Chapter 5	"Users and Authorizations": how to define the user profiles and capabilities that identify LIBRARIAN users and what they are allowed to do.
Chapter 6	"Projects": how to set up and maintain projects with LIBRARIAN.
Chapter 7	"Versions": discusses the LIBRARIAN version management and forward versioning capabilities.
Chapter 8	"Reports": how to generate reports and online inquiries; description of **SHOWLOG** commands.
Chapter 9	"Housekeeping": periodic operations to keep LIBRARIAN running efficiently.

Appendix A	"Automatic Decompression": how to automatically decompress files when users access them.
Appendix B	"LIBRARIAN Utility Program": how to use the LIBRARIAN Utility program (**LIBUTILP**).
Appendix C	"LIBRARIAN Configuration Program": how to use the LIBRARIAN Configuration program (**CONFIGP**).
Glossary	A Glossary of Terms is provided at the back of this guide.
Index	An index of LIBRARIAN topics at the end of this guide.

Conventions

We use the following conventions throughout this guide.

COMMANDS All commands appear in bold capital letters. If a command can be abbreviated, the optional portion of the command is enclosed in brackets ([ ]). A blank space *must* separate the command from the parameter list.

KEYWORDS Keywords and parameters (shown in bold capital letters) must be entered exactly as specified.

italics Words or characters in italics represent variables or arguments that you must replace with an actual value. In the following example, you must replace *fileset* with the name of the file you want to copy.

```
>COPY fileset
```

Italics are also used to introduce new terminology or for emphasis.

punctuation Enter punctuation exactly as shown. (Refer to specific instructions for brackets and braces, below.)

{ } Braces enclose required elements. When there are several elements within braces, you must select one element. In the following example, you must select one of **PROCEDURES**, **PROJECTS**, or **STEPS**.

```
>HELP { PROCEDURES }
      { PROJECTS   }
      { STEPS      }
```

[ ] Brackets enclose optional elements. In the following example, brackets around the letters UPDATE indicate that you do not have to type the entire word.

```
>AUTO[UPDATE]
```

If there are several elements, you can select any one or none of them. In the following example you can select **BATCH, CONFIRM** or **MEMO**, or none.

```
>COMPRESS [ filelist ]
      [ ;BATCH ]
      [ ;CONFIRM ]
      [ ;MEMO ]
```

When brackets are used, you cannot enter a value in the inner brackets unless you enter a value (wildcard or literal) in the outer brackets.

...	An ellipsis indicates that the previous bracketed element can be repeated or that elements have been omitted.
&	An ampersand indicates that the command continues on the next line.
MPE	The white flag symbol indicates that the text pertains to LIBRARIAN running under the MPE operating system.
UNIX	The gray flag symbol indicates that the text pertains to LIBRARIAN running under the UNIX operating system.
iX+	The striped flag symbol indicates that the feature being described is only available with LIBRARIAN/iX–Plus.
MPE only	This symbol identifies LIBRARIAN commands that have no equivalent under the UNIX operating system.

File Naming Conventions

In specifying files, LIBRARIAN commands use the following wildcard conventions:

MPE	@	Zero or more alphabetic and/or numeric characters. Used alone, denotes all members of a set.
UNIX	*	Zero or more alphabetic and/or numeric characters. Used alone, denotes all members of a set.
MPE	#	Single numeric character.
	?	Single alphabetic or numeric character.

In addition, a slash (/), a single period and slash (./), a double period and slash (../), or a tilde and a slash (~/) immediately preceding a filename indicate a UNIX file.

Related Documentation

Along with this manual, you can refer to the following documentation by OCS.

The *LIBRARIAN/iX Reference Guide* provides information on LIBRARIAN functions, including complete command syntax and reference material for all LIBRARIAN features.

The *LIBRARIAN/iX User's Guide* contains instructions on how to perform LIBRARIAN functions. It also includes detailed descriptions of the MAKE facility and the XEQ macro language.

Online help contains the contents of all LIBRARIAN manuals. You can access online help with the **HELP** command or pressing **F1 (Help)** in menu mode.

Client Services

LIBRARIAN is supported by OCS Client Services, which is dedicated to providing timely and accurate information and solutions. For fast, accurate answers, we maintain a telephone hotline that includes emergency after-hours service. You can count on OCS to isolate any problems quickly and provide conscientious support and a fast response.

Operations Control Systems hotline numbers:
Phone (415) 493-4122
FAX (415) 493-3393

Your Comments

We value your comments. As we write, revise, and evaluate our documentation, your opinions are the most important input we receive. Please use the Reader's Comment Form at the end of this guide to tell us what you like and dislike about and of the OCS manuals.

Introduction 1

LIBRARIAN is easy to implement, learn, and use. Because the needs of companies and organizations change over time, LIBRARIAN encompasses functionality to meet the file management needs of a wide range of development environments.

This chapter will give you an overview of typical file management policies and objectives and will show you how LIBRARIAN can be configured to handle a few different file management scenarios.

Topics discussed in this chapter include:

- File Management Objectives
- File Management Rules
- Examples of File Management Rules

Note If you are already familiar with basic LIBRARIAN concepts and know how to design file management policies and procedures, you can use this chapter as a checklist to help you implement LIBRARIAN, or you can skip this chapter and go straight to Chapter 2, "Getting Started".

File Management Objectives

File management objectives vary considerably between organizations. Some MIS shops allow programmers to perform their own testing and to move the tested files into production; others require independent testing and move to production. Some organizations incorporate a policy in which all changes are authorized and tested; others require absolute prevention of unauthorized changes. Some auditors require positive evidence and periodic verification of source/object synchronization; others accept reasonable assurance through procedural controls.

LIBRARIAN is designed to accommodate these differences in file management objectives. With LIBRARIAN, you can manage the development cycle rigidly or loosely, depending on your needs. Because of this flexibility, it is essential to identify and define your objectives before you begin to implement the product.

The following section lists some typical file management objectives. Use this list as a starting point for evaluating your own objectives. This list is by no means exhaustive. Create a list of your own file management objectives so you can refer to them as you define your file management rules in the LIBRARIAN database.

Typical File Management Objectives

LIBRARIAN provides a robust set of features that allow you to achieve a wide range of objectives. Table 1-1 lists typical file management objectives and the corresponding LIBRARIAN features that help you achieve the objectives.

Table 1-1 LIBRARIAN Features Related to Objectives

File Management Objective	Corresponding LIBRARIAN Features
1. Improve efficiency and convenience	LIBRARIAN provides mass file movements, customized file movement commands, file push movements across accounts and system boundaries, complete audit trail, and automated maintenance.
2. Control copies of source, object, jobstreams, etc.	Define files in a master library.
3. Require approval of changes	Define CHECKIN step requiring an approval prestep. Authorize specific users to perform the APPROVE step.
4. Require testing of changes	Define rules requiring a step to document testing before allowing checkin.
5. Prevent overlapping changes	Assign serial access control to files.
6. Synchronize source/object	Use MAKE to compile changed source automatically, use VERIFY option on checkin and file distribution.
7. Enforce separation of duties	Authorize different users to perform specific steps.
8. Require independent testing	Authorize specific users to perform testing.
9. Restrict access to master files based on application	Authorize programmers only for steps within specific applications.
10. Associate work with project or service request	Require project codes for all steps in the route; authorize programmers for specific projects.
11. Maintain backup copies of old versions	Use "retention" feature on checkin.
12. Control versions on remote systems	Use LIBRARIAN to distribute software; audit trail tracks versions.
13. Provide audit trails	LIBRARIAN automatically maintains an audit trail of all file movements.
14. Review specific file changes	Use **L/COMPARE** and **PRINT ;ANNOTATE** to compare file versions and display differences.
15. Maintain current release while developing next release	Use "branch and merge"; consider separate maintenance and development routes.
16. Maintain concurrent revisions of individual programs	Use the revision control facility.
17. Tracking versions	Use LIBRARIAN version stamping facilities.

File Management Rules

Once you have identified your file management objectives, as described in the preceding section, you will be ready to define file management rules to achieve those objectives.

Begin by identifying the files for the library and defining how, where, and by whom files are moved in the process of making software changes for a single application.

Master Library Files

The master library for an application includes all of the files that LIBRARIAN manages for that application. These files are called *master files*. Master files are the official or control copies of files to be managed. Copies of master files in other locations are tracked by LIBRARIAN as *secondary copies*. Your file movement rules define how copies of master files are checked out and modified, and how they eventually replace the master files in the library through controlled file movements. For detailed information refer to Chapter 3, "Master Library".

Diagram File Movements

Decide how and where changes are made to files within the application. Identify all of the processes or cycles for the application. These cycles are defined to LIBRARIAN as *routes*. For example, you could identify routes for maintenance, for development, for distributing data, etc. You could have alternate routes for a given group of files, depending on whether the file is accessed for maintenance or development, or whether a correction is a routine or emergency procedure.

Next, identify the specific file movements that comprise the cycle for each route. For example, most software maintenance cycles begin by copying the master file to a development location. Then, the files are moved to a secured area for testing. Most maintenance cycles end by moving the modified file(s) back into the master library. Each of these file movements is called a *step*. Steps are defined in the rules database.

Some steps require previous approval by a manager, or sign off by a user or tester. These approvals and sign offs are also considered steps in LIBRARIAN.

To help visualize the sequence of file movements and approvals in the cycle, we recommend drawing a flow diagram showing the sequence of steps and the various locations that are used for modification testing. Several examples of this diagramming technique are included later in this chapter (see Figure 1–1).

Identify Rules for File Movements

Once you have identified and mapped out the steps and routes for your development cycle, you are ready to identify rules for performing individual steps. To do this, first define *what* files can be moved, *how* the files should be moved, *who* can move the files, and what external events are required (*when*). The following list highlights examples of LIBRARIAN rules you can apply to a step:

- **Scope Restriction** — Limit the file movement to a subset of the application's files, or to files in a particular location.
- **Destination Restriction** —Define the destination location of moved or copied files performed by this operation.
- **Copy, Move, or Null Operation** — Use
  - COPY to copy, but *not* purge the original files,
  - MOVE to copy *and* purge the original files,
  - NULL to record an event, but not physically move files.
- **Prerequisites** — Define one or more steps that must be performed on a file prior to performing this step (often, but not necessarily, an approval step).
- **Retain Backup Copies** — Rename existing destination files for backup prior to the move or copy operations or, during checkin, retain saved changes in a delta file.
- **Allow New Files to be Introduced** — Enforce naming conventions.
- **Authorized Users** — Identify specific users who can perform a step.
- **Read or Write Copy** — Only copies with write mode access can replace the original in the master library.
- **Require Project Identification** — You can associate all file activities with projects (applies to all steps in a route).

This is only a partial list of step considerations. Many other features or operations are possible by combining LIBRARIAN options. As you become more familiar with LIBRARIAN, you will find more features that you want to implement. The examples in this chapter illustrate how some LIBRARIAN features are implemented.

Examples of File Management Rules

It is impossible to define a single, ideal set of file management rules for every installation, but the following four examples represent typical implementations of LIBRARIAN.

The first example depicts the most basic environment — a typical development cycle in a small MIS shop. The remaining examples show more complex implementations of LIBRARIAN. As you review these examples, keep in mind that they are intended only as representative examples. You can customize LIBRARIAN to meet your own unique requirements.

Example 1 — Basic Development Environment

This example illustrates a shop with a single system and a staff of three programmers, one day-shift operator, and a manager. The sample environment is a development route in a typical application. The shop's file management objectives for this route include:

- tracking the official copies of all source, object, and job files;
- ensuring that all changes are made to *copies* of those files in a separate development location;
- ensuring that all changes are approved by the manager before release; and
- avoiding adding staff or further burdening the operator.

First, identify all of the current production source, object, and job files. These files are secured as the master library. Within the library, group the files into site-defined logical *filesets*. There is no need to change the MPE/HP-UX directory structure or move files. You can secure files using normal MPE/HP-UX security.

Next, define the file movement steps for the route. In this example, the steps for the route are defined as follows:

1. The CHECKOUT step moves source, object, or job files from the master library into a development location.
2. The APPROVE step allows managers to record their approval on a file-by-file basis.
3. The CHECKIN step returns approved files to the master library and automatically creates a backup copy of the old master file(s).

Figure 1–1 shows a flow diagram for this route using these three steps.

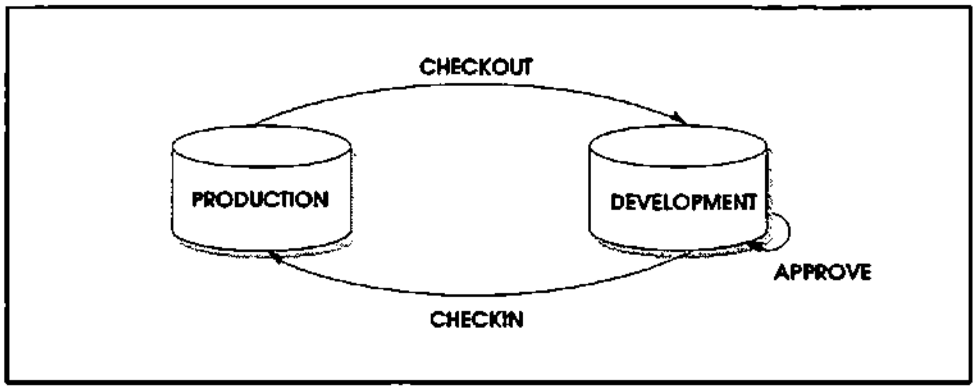


Figure 1–1. Basic Development Route

You can define the CHECKOUT step to create test copies of the master files even though the master library is secured. LIBRARIAN keeps track of these copies and can prevent multiple programmers from checking out copies of the same file simultaneously.

You can define the APPROVE step to mark the file(s) as approved, and to restrict use of this step to the manager only.

You can define the CHECKIN step to allow programmers to push approved copies from the test area into the master library, without logging in to the master location. A backup copy of the old master file can be archived automatically on disk, and it can be compressed automatically to save disk space.

Example 2 — Separate QA Function

The second example illustrates a development route for a medium sized shop with tightly controlled procedures. The staff includes a manager, six programmers, two operators, and a quality assurance (QA) analyst. Due to the complex nature of their applications, this shop insists on a formal quality assurance process for every change, and management approval of the results before any new or changed software can be incorporated into production.

However, this shop also needs to allow for quick emergency fixes by programmers without the delays that normally accompany the QA process. Furthermore, a complete and reliable audit trail must be maintained for these quick fixes.

Define the file movement steps for the route. In this example, the steps for this route are defined as follows:

1. The route begins with the CHECKOUT step, as defined in Example 1. The master files are secured, but programmers can copy files easily to the development area for modification. The master files have serial access control, permitting only one copy at a time to be modified to replace the master file. This prevents lost work due to overlapping changes. Other copies are available with read–mode access.
2. The SUBMIT step is used by the programmers to move modified source, object, and job files into the QA area for testing.
3. The GET-CURRENT step copies the latest versions of files from the master library to the QA area in read–mode. This step keeps the QA environment up–to–date with a complete set of software.
4. The REJECT step is used by the QA analyst when program testing fails. It moves rejected files back to the development area.
5. The TESTOK step is used by the QA analyst to indicate that a change passed system tests. This step helps the manager maintain the status of work on various programs.
6. In this example, the APPROVE step operates on files in the QA location. The APPROVE step can be used only after the file passes the TESTOK step.
7. The CHECKIN step moves approved files from QA to the master library.

8. The FIX step moves files directly from the development area to the master library for emergency fixes. This step is configured with the **PUSHREAD** option, enabling it to be used if a write-mode copy of the file exists previously when checked out. When this step is executed, any write-mode copies are flagged so the user is warned of an overlapping change the next time the file is moved.

Figure 1–2 shows a flow diagram for this route using these eight steps.

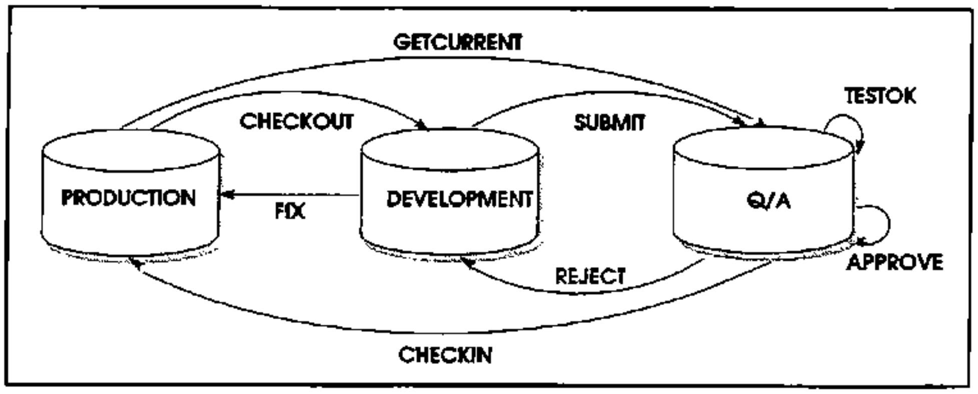


Figure 1-2. Separate QA Route

LIBRARIAN uses steps to accomplish separation of duties. The steps operate on logical groups of files, enabling users to move or copy groups of files without specifying file names.

LIBRARIAN automatically maintains a complete audit trail. Therefore, no additional effort is required to monitor emergency fixes. The manager simply runs a SHOWLOG report listing all uses of the FIX step.

Example 3 — Networking and Source/Object Synchronization

This example illustrates the use of separate computers for production and development. Many applications run on remote systems. The master library contains the production source, object, and jobstreams, but the executable object and jobs are distributed to the appropriate remote systems. An important objective is ensuring that all of the remote sites are running the same code; object files in the master library and on all remote systems must be generated from the correct source files in the master library.

In this example, the steps for this route are the same as Example 2, with the following exceptions:

1. A new step, called RELEASE, is defined to distribute object and job files to the remote sites. This step copies files to all remote sites and produces an audit trail to verify that the copies were successful.
2. A macro called **SUBMIT** invokes the MAKE utility to recompile programs automatically as they are moved into the QA area, ensuring source/object synchronization at that point.

3. The CHECKIN step is configured to require that the source and object files are moved together, and that files cannot be checked in if their timestamps have changed since TEST-OK approval was done in the QA area.
4. The remainder of the steps function as they did in Example 2, except that they have been defined to work in a network. The CHECKOUT, GET-CURRENT, CHECKIN, and FIX steps operate between the production and development machines. The RELEASE step operates between the production system and the remote sites.

With LIBRARIAN, you can implement these additional steps with minimal effort; they can even be performed in batch jobs.

Figure 1–3 shows a flow diagram for the route using these steps.

Automatic recompiling of source for source/object synchronization is accomplished by running MAKE regularly to automatically create and stream compile jobs for source files that moved into the test area. No special steps are required to accommodate the dual–machine development environment or the remote sites. LIBRARIAN allows steps to be defined across machines. The result is transparent to the user.

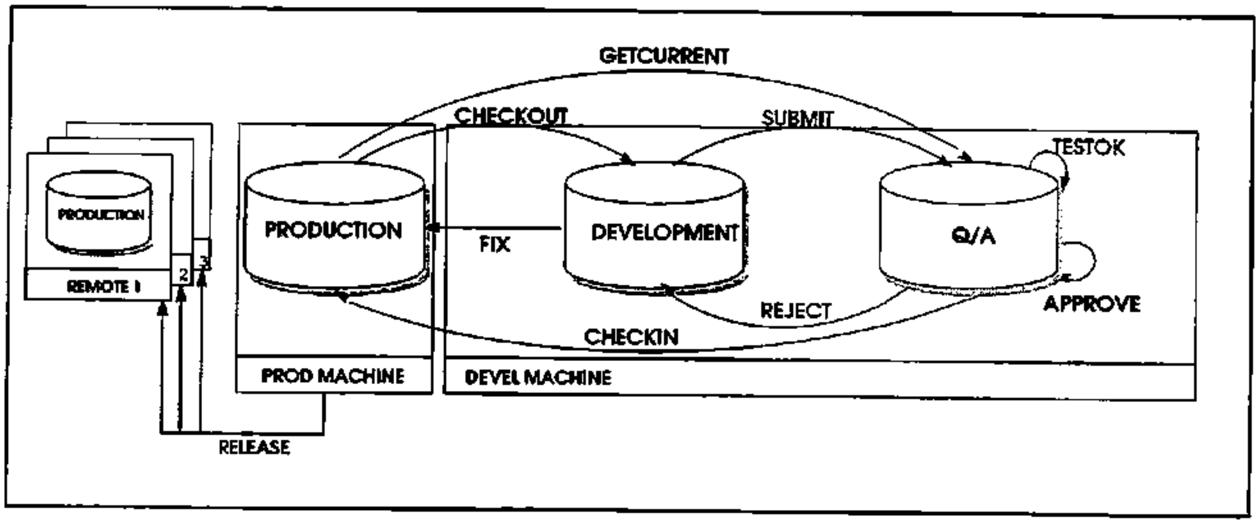


Figure 1-3. Network Route

Example 4 — Multiple Version Management

This example illustrates an application with two routes used concurrently: one route for a long–term development project, and one route for maintaining the current version during the development period.

In this example, the steps for the routes are defined as follows:

1. The maintenance route for the current version (2.0) has five steps. CHECKOUT2, SUBMIT2, REJECT2, APPROVE2, and CHECKIN2 are the same steps as the CHECKOUT, SUBMIT, REJECT, APPROVE, and CHECKIN steps described in Example 2.
2. The development route for the new release (3.0) includes five steps. The SUBMIT3, REJECT3, and APPROVE3 steps are the same as the SUBMIT, REJECT, and APPROVE steps described in Example 2.
3. The CHECKOUT3 step selects the latest version of the software from the master library. The step is defined with a primary search location and an alternate search location. If the file to be checked out is not found in the 3.0 library, it is retrieved from the 2.0 location.
4. The CHECKIN3 step replaces files in the release 3.0 master library if they were checked out of 3.0, or introduces new files to the 3.0 master library if they were checked out of release 2.0. This cycle ensures that programmers do not work on the old versions of files if files have already been modified for the new version.

Figure 1–4 shows a flow diagram for the routes containing the steps listed above.

This development route implements the *forward versioning* feature. Defined file movements search through two or more alternate locations for the current version of a file.

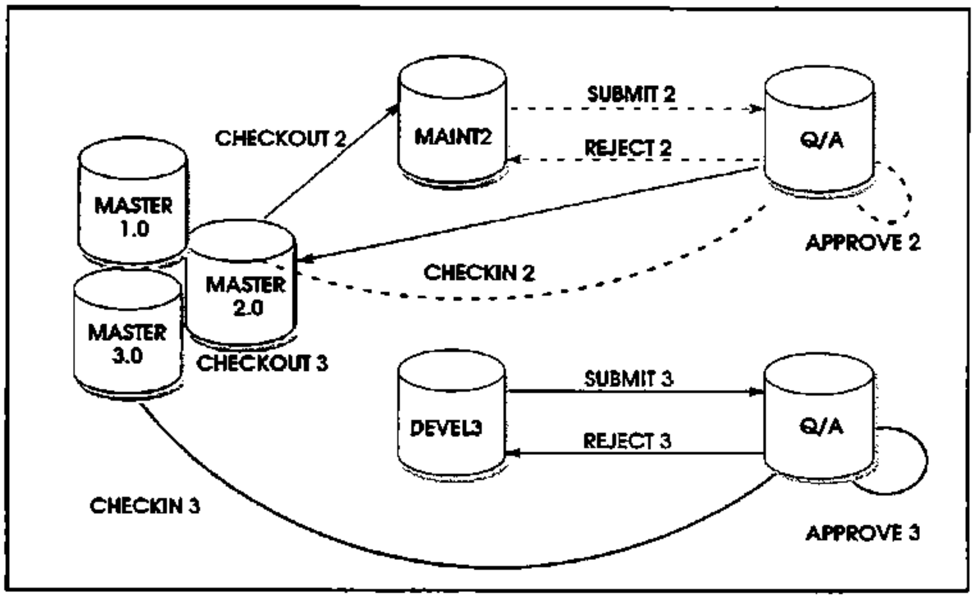


Figure 1–4. Version Control Routes

Notice that LIBRARIAN allows you to manage this rather complicated, but not uncommon, environment with minimal effort. With forward versioning, you can maintain the integrity of prior versions while building new versions, without replicating files that did not change. Users always get the correct version of a file for maintenance or development.

Getting Started with Basic Rules 2

After installing LIBRARIAN, you are ready to define a change control cycle and begin controlling and recording file movements. This chapter provides information on how to define a basic set of rules using the Shortcut utility.

Topics in this chapter include:

- Basic Setup
- Deleting an Application
- Customizing the System Profile
- Creating a LIBRARIAN Manager
- What Next?

Note Before proceeding, you need to install LIBRARIAN on all licensed systems by using the installation instructions provided with the product tape.

Basic Setup

The Shortcut utility is designed to help you quickly implement the basic LIBRARIAN features, including checkout/checkin, serial access control, file retention and compression, audit trails, and more. After running Shortcut, you can use the LIBRARIAN screens to fine-tune the basic environment, and take advantage of additional LIBRARIAN features.

With Shortcut, you will be able to start using LIBRARIAN immediately. After answering a few questions about your environment, the program will ask if you want to proceed with loading this data into the LIBRARIAN database. Once the data is loaded, you can begin to use LIBRARIAN. It enables you to move and copy files, control changes to files, and provide extensive audit reporting.

The entire process of setting up a LIBRARIAN implementation for an application (i.e., a set of logically related files and movement rules) takes only a few minutes with LIBRARIAN's Shortcut.

Running Shortcut

To run Shortcut, login and run LIBRARIAN.

MPE

To access LIBRARIAN from MPE, type:

```
:LIB
```

UNIX

To access LIBRARIAN from UNIX, type:

```
HP-UX[1] ocslib  if path is set,
```

otherwise

```
HP-UX[1] $OCSLIBDIR/ocslib
```

where $OCSLIBDIR is the name of the directory where the LIBRARIAN client software is installed.

Enter LIBMGR when prompted for a user ID. If this is the first time you are using LIBRARIAN, you will be prompted to assign a password. Please note that user IDs and passwords are case–sensitive in LIBRARIAN.

Note LIBMGR is the only user defined to LIBRARIAN at the time of installation. See Chapter 5, "Users and Authorizations", for a discussion of how to add new users and assign special capabilities.

The LIBRARIAN Main menu appears after you successfully provide a user name and password. Select **Admin** from the menu bar and select **Shortcut** from the pull-down menu (see Figure 2–1).

Shortcut gives you explanations at every step and provides you with additional online help.

Note If you are using Shortcut for the first time, we recommend that you press F1 for each instructional window during the Shortcut dialog.

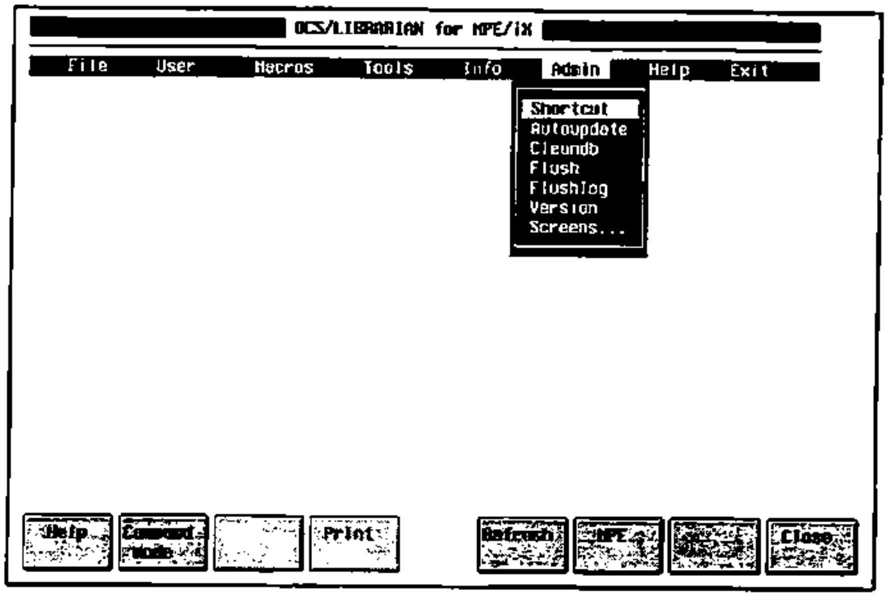


Figure 2–1 Admin Menu from the LIBRARIAN Main Menu

Function keys defined for Shortcut are described below in Table 2–1.

Table 2–1 Shortcut Function Keys

Key	Function
Return	Accepts default answer to a prompt shown in [brackets].
F1 Help	Displays context-sensitive help.
F2 Restart part	Restarts from beginning of current part (available only when function key label is displayed).
F3 Previous part	Returns to beginning of previous part (available only when function key label is displayed).
F4 Refer to	Displays location of additional information.
F6 Refresh	Refreshes the display.
F7 Start over	Starts over from beginning of Shortcut.
F8 Exit	Terminates program without saving data.

Shortcut will guide you through setting up change control for an application, as summarized below.

Identify An Application

Applications provide the organizational framework for file management with LIBRARIAN. They let you organize your files into separate master libraries, and define different change control rules and versions for those files. Before using Shortcut, select an application that you would like to start with.

Create a one- to four-letter Application ID that you will later associate with a library of files, along with a set of file movement commands. For example, if you are working with a payroll application, the ID might be PAY.

Define the Application Library

A library is a collection of files that you want to control for an application. To identify the files that make up your library, use the following format:

MPE *system:file.group.account*

UNIX *system:/path/file*

You can use wildcards.

Define Steps

Shortcut automatically creates file movement and approval commands called *steps*. First, you need to define the destination location for a checkout step by selecting an option from the menu that Shortcut displays. You can then choose additional steps you need to copy/move files and/or record approvals.

The names of the steps you choose will have the application ID as a prefix and a dash followed by an abbreviated step name. For example, a payroll application with (PAY) might include the following steps:

PAY–OUT	Checkout files to development
PAY–NEW	Introduce new files from development
PAY–IN	Checkin files to library
*PAY–OK	Approve files
*PAY–TESTOK	Approve files to be checked in from test area
*PAY–TEST	Move files from development to test area
*PAY–FAIL	Move files back to development from test area
*PAY–READ	Copy files in read–mode

(*optional steps)

You can add, modify, or delete the steps at any time using the Steps (ST) screen which you access from the **Screens...Steps** from the **Admin** menu.

Identify Users

Identify the users authorized to use LIBRARIAN, and the specific steps each user is permitted to perform.

Review and Load Data

You will now be given an opportunity to review the information you just provided. You can then instruct Shortcut to load this data into the LIBRARIAN database (or exit without saving).

Troubleshooting

If a connection error occurs when the library is being loaded with MPE files, it means that LIBRARIAN could not link to a remote system that you specified when you identified your library. But don't worry, you won't need to start over. The most frequent reason for a network linking problem is that LIBRARIAN is not correctly installed on the remote machine.

Verify it by checking the $STDLIST files for the installation jobs, or try installing again. Logon security may also cause a linking error. Check remote passwords on the NC screen. If you are using third–party security software on a remote system, the LIBRARIAN logon may have to be configured in that software's database. A less common reason has to do with the configured buffer size for X.25 networks. If you are using a X.25, you may need to change the default buffer size to 138 words on the Network Configuration (NC) screen in LIBRARIAN.

If you can correct the problem, try loading the library again with the **AUTOUPDATE** command or the **AUTOUPDATE** option from the Admin menu. If a failure still occurs, contact OCS for assistance.

If a connection error occurs when the library is being loaded with UNIX files, make sure that LIBRARIAN has been installed properly on the UNIX system.

Deleting an Application

If the application you set up with Shortcut is not what you want, or you created an application for evaluation purposes and have completed the evaluation, you can easily delete your library definitions and file movement rules from the database. Use the following steps to delete an application:

1. Invoke LIBRARIAN, as mentioned in the beginning of this chapter.
2. From the LIBRARIAN prompt, type:

   ```
   DELETE application
   ```

   where *application* is the application ID you specified in Shortcut. You will be prompted to confirm the deletion.

Setting Password Security

The LIBRARIAN Manager can specify the following password requirements for new and current users. Use the SP screen to specify these password requirements, as indicated below:

Aging (Days Valid)
Required. Length 3.

The number of days passwords are valid.

Passwords can expire between 1 and 500 days. The default expiration period is 999 days, indicating that user passwords will not expire.

Minimum length
Required. Length 8.

The minimum number of characters required for each LIBRARIAN user password.

You can set passwords to have a minimum length of 1 to 8 characters. The default minimum password length is 1 character.

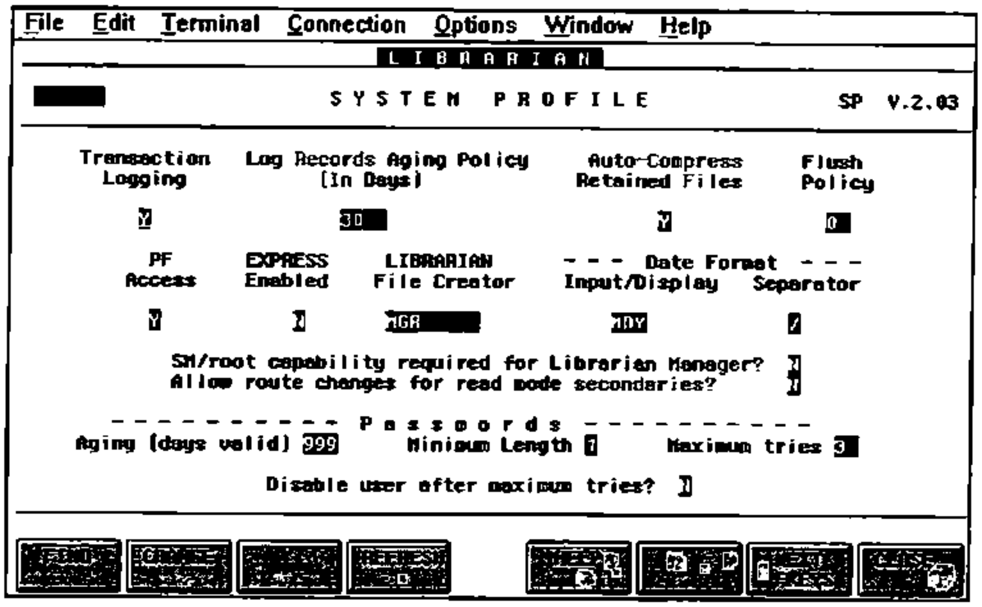
Maximum tries
Required. Length 2.

The maximum number of times a user can attempt to enter a valid password.

If a user exceeds this number of attempts and the "Disable user after maximum tries?" field is set to "Y", LIBRARIAN will disable the user ID. The LIBRARIAN Manager will then have to re-activate the user with on US screen.

You can set the maximum number of attempts to a value between 1 and 16. The default is 3 attempts.

Disable user after maximum tries?
If this field is set to "Y", when the user exceeds the maximum number of attempts to enter a valid password, LIBRARIAN will disable the user ID by setting the active flag to "N". The LIBRARIAN Manager can re-activate the user on the US screen.



```
                              L I B R A R I A N

                      S Y S T E M   P R O F I L E                SP  V.2.03

  Transaction   Log Records Aging Policy    Auto-Compress     Flush
    Logging           (In Days)            Retained Files     Policy

       Y                  30                     Y              0

      PF        EXPRESS     LIBRARIAN      - - -  Date Format  - - -
    Access      Enabled    File Creator    Input/Display   Separator

       Y           N         MGR               MDY             /

          SM/root capability required for Librarian Manager?  N
          Allow route changes for read mode secondaries?      N

      - - - - - - - - - - -  P a s s w o r d s  - - - - - - - - - - -
  Aging (days valid) 999     Minimum Length 7      Maximum tries 3

                  Disable user after maximum tries?  N
```

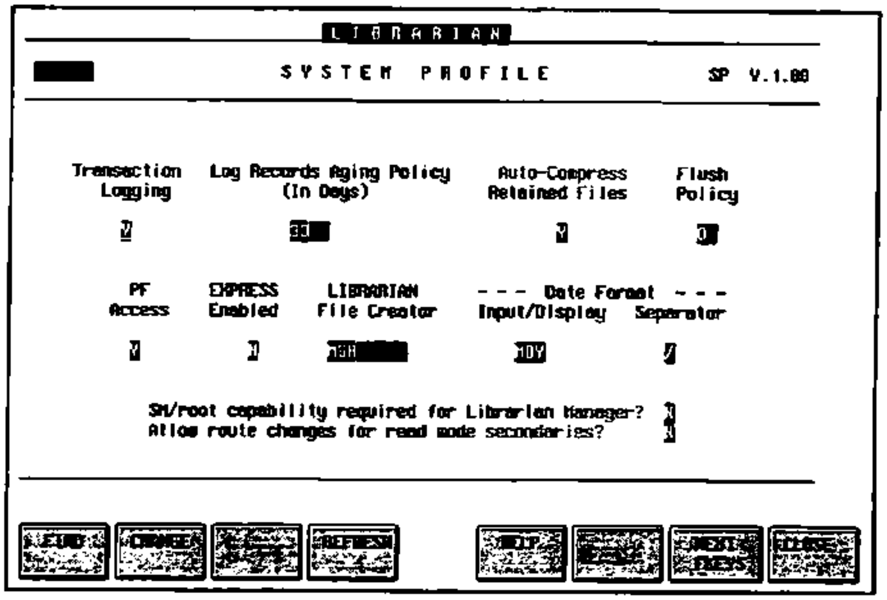
Figure 2–2. System Profile (SP) Screen

Customizing the System Profile

Default parameters are defined in the System Profile when you install LIBRARIAN. These parameters affect the maintenance and audit trail logging for your entire installation. Use the System Profile (SP) screen to review and customize these parameters (see Figure 2–3). If you prefer, you can leave them as they are and customize them later.

To access the System Profile (SP) screen, type SP at the command line prompt or select **SP System Profile** from the **Admin...Screens...Config** menu.

The System Profile (SP) screen displays default values, as shown in Figure 2–3. If you require help, press **F5** (HELP) or refer to Chapter 5, "Screens", in the *LIBRARIAN/iX Reference Guide.*



```
                         L I B R A R I A N
                    S Y S T E M   P R O F I L E              SP  V.1.00

   Transaction   Log Records Aging Policy    Auto-Compress    Flush
     Logging           (In Days)            Retained Files   Policy

        Y                 30                       Y            [illegible]

     PF         EXPRESS      LIBRARIAN      - - -  Date Format  - - -
   Access       Enabled     File Creator    Input/Display   Separator

      Y            N        [illegible]          MDY             /

        SM/root capability required for Librarian Manager?  N
        Allow route changes for read mode secondaries?      N
```

Figure 2-3 System Profile (SP) Screen

Creating a LIBRARIAN Manager

The LIBRARIAN Manager is responsible for defining libraries, file movement rules, and authorizing users to perform LIBRARIAN operations. LIBRARIAN Manager capabilities are very powerful; they assign access to all data, all files, and all file movements on all systems. The LIBRARIAN Manager can perform all LIBRARIAN operations, although the LIBRARIAN Manager capability can be restricted to users with SM capability (under MPE) or root capability (under UNIX) on the System Profile (SP) screen.

LIBRARIAN is provided with a single predefined LIBRARIAN Manager user—LIBMGR. Initially, there is no password assigned to LIBMGR.

You need the LIBMGR user to access LIBRARIAN for the first time. Before you load file management rules, you can define your own LIBRARIAN Manager and delete the predefined LIBMGR user and capability data.

To create a new user with LIBRARIAN Manager capabilities, first define a user and password on the Users (US) screen. Then, assign LIBRARIAN Manager capability to that user on the User Capabilities (UC) screen.

What Next?

Now that you have described your application and defined the file movement rules to LIBRARIAN using Shortcut, you are ready to use LIBRARIAN to manage software files and control changes.

Refer to Chapter 2, "Getting Started", in the *LIBRARIAN/iX User's Guide* for information on how to use LIBRARIAN.

The remaining chapters of this guide provide details of LIBRARIAN setup to help you understand and tune the rules that Shortcut created for your environment. Advanced options are also discussed.

Master Library 3

This chapter describes master library concepts and how to define a library. Topics include:

- Defining an Application
- Defining a Library
- Defining Filesets
- Customizing File Access Rules
- Reviewing Library and File Information
- Deleting Library and File Information
- Summary

The first step in setting up an application is to define the master library. For each application, you need to identify the files that belong to the library and organize them into a hierarchy of filesets, if desired. Use the data-entry screens to define the library. If you used Shortcut to define an application. The application consists of only a single application fileset. You can use the screens discussed in this chapter to create a hierarchy of subsets for that application fileset.

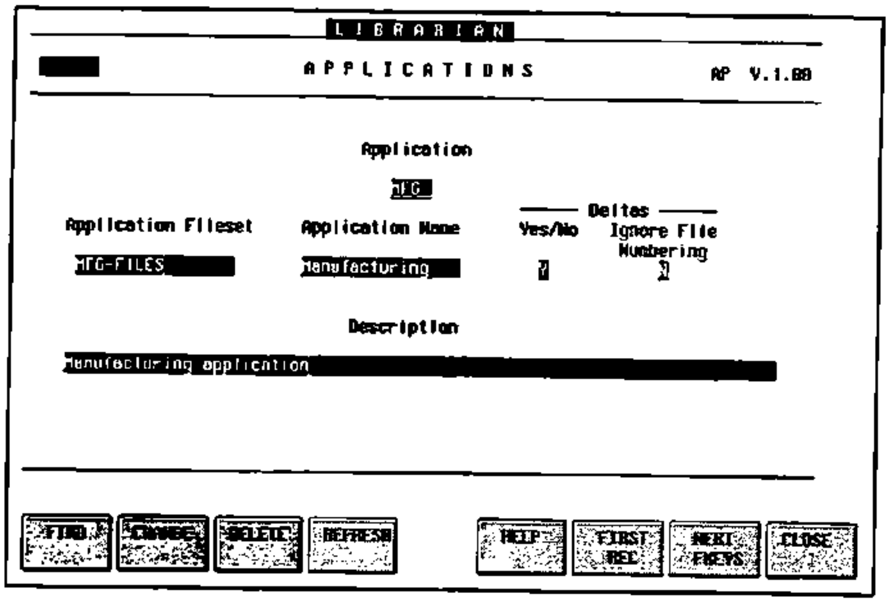
You should begin by using default values until you become more familiar with the product. Later, you can fine tune LIBRARIAN to meet your specific needs.

Defining an Application

Applications are the highest organizational unit in the LIBRARIAN system. The library master files for an application are defined as a fileset, or fileset hierarchy, as discussed below. The file movement rules of the application are associated with the library through the application name. In addition, the application's projects (if used) are associated with the application name.

Use the Applications (AP) screen to define the name of your application and associate a fileset with that application.

Figure 3–1 contains a sample application definition for MFG. The application fileset is MFG-FILES.



```
                         L I B R A R I A N
                      A P P L I C A T I O N S             AP  V.1.00

                              Application
                                 MFG
                                              ——— Deltas ———
Application Fileset   Application Name       Yes/No   Ignore File
                                                      Numbering
MFG-FILES             Manufacturing            Y          N

                              Description
Manufacturing application
```

Figure 3-1 Applications (AP) Screen

Applications (AP) Screen Fields

This section describes the fields on the AP screen, enabling you to define an application and associate your library with it.

Application	A unique identifier for the application consisting of a maximum of four characters. The application can include alphabetic, numeric, hyphen (-), and underscore (_) characters.
Application Fileset	Highest level fileset in the application. It includes all filesets and files in the application. If the fileset does not already exist, it is created automatically with serial access control and write mode default access. You can override these values later with the Filesets (FS) screen, or you can override them on a file-by-file basis with the File Access (FA) screen, as described on page 3–11.
Application Name	Provides space for a longer application name for documentation purposes.
Deltas (Yes/No)	Specifies whether to use delta files or generation files for text files.
Deltas (Ignore File Numbering)	Specifies whether file numbering is significant when determining deltas. This field is applicable only if you specify Y in the deltas Yes/No field.
Description	A description of the application for documentation purposes (optional).

Defining a Library

An application's library is the collection of all filesets associated with it. Each application must have a minimum of one fileset. The following are some of the reasons for defining multiple filesets for an application:

- *Convenience* — Filesets are groups of files likely to be used together. You can specify fileset names in LIBRARIAN commands to perform steps on groups of files. If you require subgroups, create as many levels as needed for ease of reference. Users can create their own user filesets for ad hoc file movements, as described in Chapter 6, "User Filesets," in the *LIBRARIAN/iX User's Guide.*
- *Access control* — Filesets establish default access rules for the entire group of files. Create subgroups to define different access control and default access modes.
- *File movement rules* — Steps are defined to apply to specific filesets. Create filesets for groups of files for which different rules apply.

Defining Filesets

First, you define fileset names with defaults for file members, then you create a fileset hierarchy. Once the hierarchy is defined, you can add individual files as members of each fileset.

Access Mode and Control

When defining a fileset, you specify a default access mode and an access control level that are assigned automatically to all physical files added to the fileset. These access requirements determine the type of copies users can obtain in LIBRARIAN operations. You can override these defaults later on a file-by-file basis.

Access Mode

LIBRARIAN assigns an *access mode* to every copy of a master file. A secondary copy's access mode determines if it is allowed to replace its master through a checkin step. The two access modes are:

W (Write)	Indicates that a secondary file *can* replace its master file.
R (Read)	Indicates that a secondary file *cannot* replace its master file.

Access Control

The master file's access control level determines how many read and write mode secondary copies are allowed. The following are the four access control levels:

S (Serial)	Only one secondary file at a time can have write mode access, but unlimited read mode copies are allowed. Provides protection against concurrent modifications.
X (Exclusive)	The master file cannot be copied.
R (Read)	Only read mode copies are allowed. These copies cannot replace the master file.
M (Multi-Write)	Unlimited read and write mode copies are allowed. No protection exists against concurrent modifications. Two or more programmers can work on a file. One programmer's changes could replace another programmer's changes in the master library.

Most development and maintenance environments use serial access control because it protects against simultaneous changes done by two or more people.

How Access Control and Access Mode Work

In a development route, a file is checked out from a master library with write-mode access. As the file moves through a route, this access is transferred automatically to the next destination copy, unless otherwise specified.

For example, if a CHECKOUT step assigns write-mode access, then the step which moves files to QA would pass along the write mode access. This enables the CHECKIN step to replace the master file in the QA copy. If the CHECKOUT step assigns read-mode access, then the CHECKIN step cannot be performed.

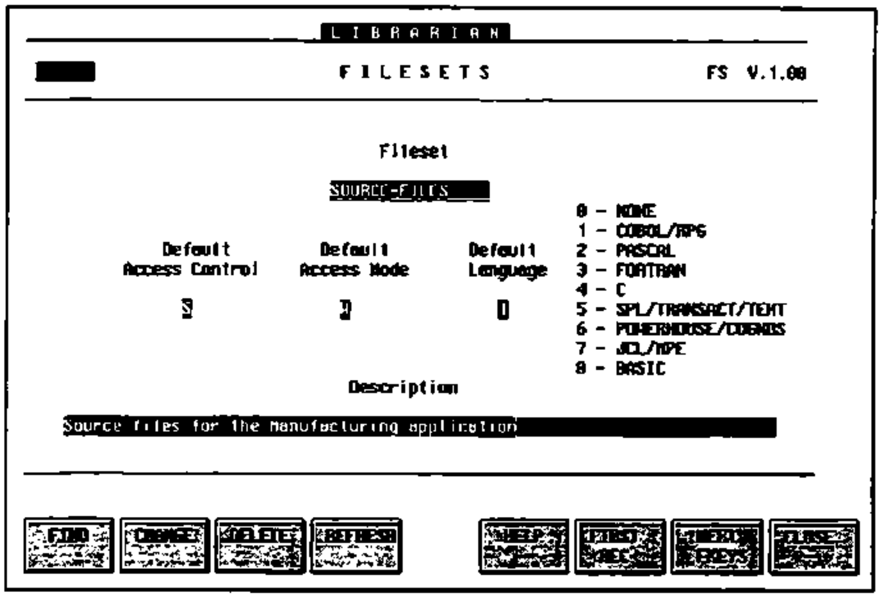
LIBRARIAN Managers and Application Managers can change a file's access mode with the **SET** function. They can also replace a master file with a read mode secondary copy by using the **PUSHREAD** parameter with a step, or the **COPY** or **MOVE** commands. In fact, an emergency step could be configured to automatically invoke this parameter for authorized users.

The following is an example of how to use access mode and access control in LIBRARIAN:

1. The ABC file has serial (S) access control and write mode (W) default access. There are no secondaries of ABC at this time.
2. The user checks out a copy of the ABC file without specifying read mode or write mode. LIBRARIAN automatically assigns write mode to the copy.
3. The secondary is copied to a test location. The command does not specify access mode, but LIBRARIAN automatically transfers write mode to the secondary in its new location.
4. A second user tries to check out a write mode copy of ABC. LIBRARIAN does not allow this additional copy, but the user can obtain a read mode copy.

Defining Filesets

Use the Filesets (FS) screen to create filesets by defining the logical fileset name and the default access parameters for fileset members. Figure 3–2 shows the fileset definition for a fileset called SOURCE-FILES.



```
                          L I B R A R I A N
                          F I L E S E T S                    FS  V.1.00

                               Fileset
                            SOURCE-FILES
                                                      0 - NONE
                                                      1 - COBOL/RPG
        Default           Default        Default      2 - PASCAL
     Access Control     Access Mode     Language      3 - FORTRAN
                                                      4 - C
           S                 W              0         5 - SPL/TRANSACT/TEXT
                                                      6 - POWERHOUSE/COGNOS
                                                      7 - JCL/MPE
                                                      8 - BASIC
                             Description
  Source files for the manufacturing application
```

Figure 3–2. Filesets (FS) Screen

Filesets (FS) Screen Fields

This section describes the fields on the FS screen for defining a fileset.

Fileset	A unique name consisting of a maximum of 16 characters. Do not use a fileset name that was previously defined for another application. Some sample fileset names are: AP-OBJECT SALES-SOURCE PAYROLL CHECK-WRITING
Default Access Control	The access control level to automatically assign files added to the fileset.
Default Access Mode	The access mode to automatically assign files added to this fileset.
Default Language	The default language to automatically assign files added to the fileset. Language controls source code connotation.
Description	A description of the fileset.

Defining a Fileset Hierarchy

Filesets are the logical components of your library. You can also define hierarchical relationships between these components. The number of levels in the hierarchy depends on your own needs.

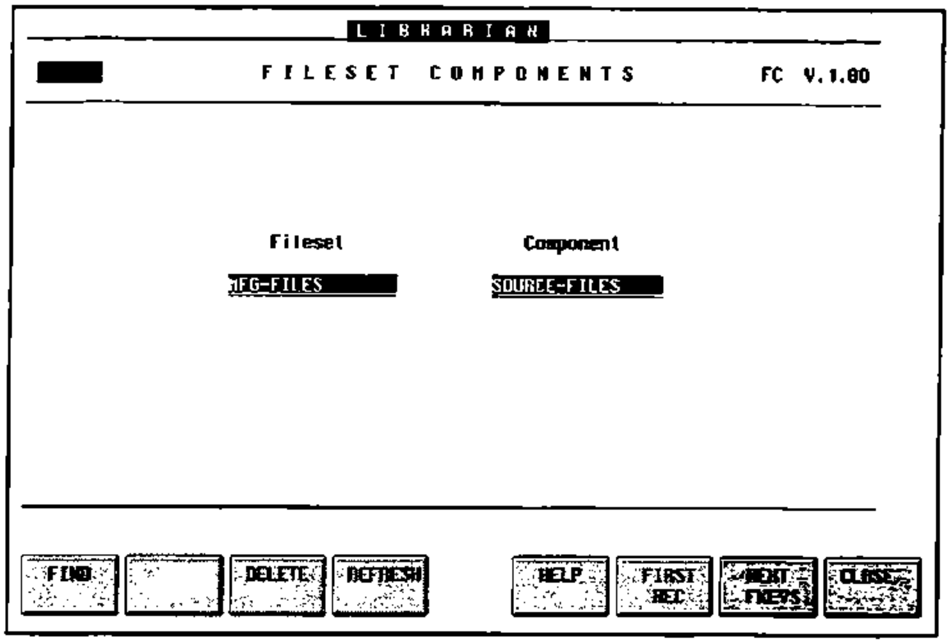
Begin with the highest level fileset, which is the application fileset. If you draw a tree diagram for your library, begin with the application fileset at the top of the tree.

Use the Fileset Components (FC) screen to define the fileset hierarchy. Enter the name of a fileset and the name of one of its component filesets. Figure 3–3 illustrates the definition of SOURCE-FILES as a component of MFG-FILES.

Fileset Components (FC) Screen Fields

This section describes the fields on the FC screen for defining your fileset hierarchy.

Fileset	A logical fileset name previously defined on the Filesets (FS) screen.
Component	A fileset to be treated as a subset of the fileset defined in the Fileset field above.



FILESET COMPONENTS FC V.1.00

Fileset: MFG-FILES

Component: SOURCE-FILES

Figure 3-3. Fileset Components (FC) Screen

Defining Physical File Members

Next, add files to the logical filesets.

Logical fileset organization is independent of MPE or UNIX directory structures and system boundaries. A fileset can include any collection of files, regardless of the file type or location. This ability to group files logically provides flexibility in creating file structures that reflect your specific needs. Create filesets for subgroups you want to manage as a unit, or that have something in common.

For example, you could create a fileset that includes these source files from various groups, accounts, and systems for an accounts payable application:

```
SYSA:AP@S.JOB.PROD
SYSA:SP@S.SOURCE1.PROD
SYSB:SP@S.SOURCE1.PROD
SYSB:LISTAP.SFILE.COMP
SYSB:APFILO.SOURCE.AP
SYSB:GLFIL##.SOURCE.AP
```

```
sysa:/prod/job/ap*s
sysa:/prod/source1/sp*s
sysb:/prod/source1/sp*s
sysb:/comp/sfile/listap
sysb:/ap/source/apfilo
sysb:/ap/source/glfil[0–9][0–9]
```

Another fileset could include the object code corresponding to these source files. Or, you might want the source and object files together in a single fileset.

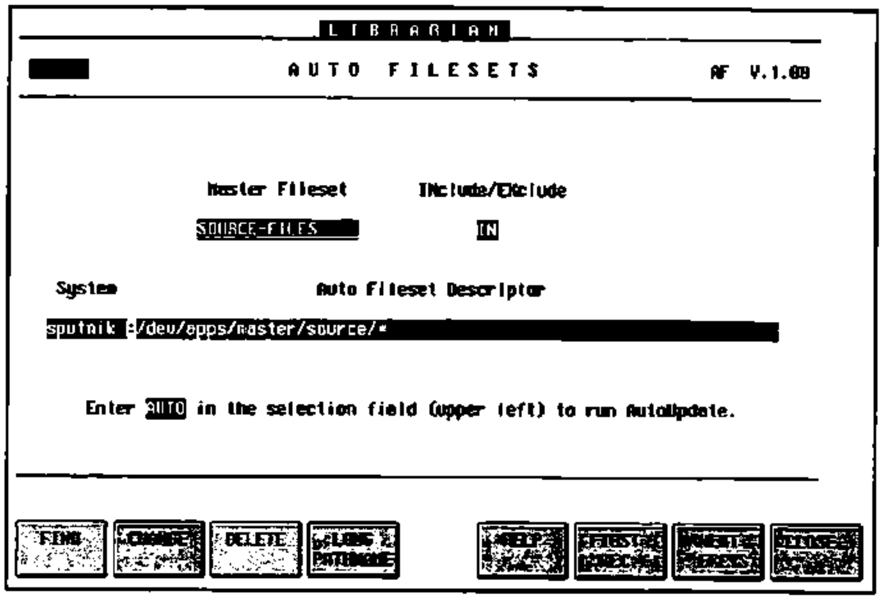
Use the Auto Filesets (AF) screen to identify the general location of files that belong to a fileset. By using wildcards in the descriptor, you can load multiple files at the same time. You can use the INclude/EXclude field to refine your fileset definition.

Note

Auto filesets are not only used to initially load your library with files, they are also used later by LIBRARIAN when new files are introduced to determine the filesets to which these new files should belong.

To add files that cannot be described by a single wildcard descriptor, you can add any number of different descriptors to identify the different file locations. Later, you can use the Auto Fileset Update (**AUTOUPDP**) program to add previously unknown files to the fileset, based on the saved auto fileset descriptors (see "Using AUTOUPDATE for New Master Files", later in this chapter.). As new master files are introduced through checkin steps with the **AUTOUPDATE** function, they are added to the appropriate filesets automatically. Type AUTO in the upper–left box (ULB) on the AF screen to run **AUTOUPDATE** immediately.

Figure 3–4 shows the use of the AF screen to define the location of files that belong to the SOURCE-FILES fileset.



```
                         L I B R A R I A N
              A U T O   F I L E S E T S                 AF  V.1.00

               Master Fileset      INclude/EXclude
               SOURCE-FILES              IN

  System                 Auto Fileset Descriptor
  sputnik :/dev/apps/master/source/*

    Enter AUTO in the selection field (upper left) to run AutoUpdate.
```

Figure 3–4. Auto Filesets (AF) Screen

Note

Remember: You are only defining master files. Secondary files are always linked to a master in the library, and are indirect numbers of filesets to which the associated master belongs. To link secondary files already in progress prior to installing LIBRARIAN to newly defined masters, refer to Appendix A, "Applications in Progress".

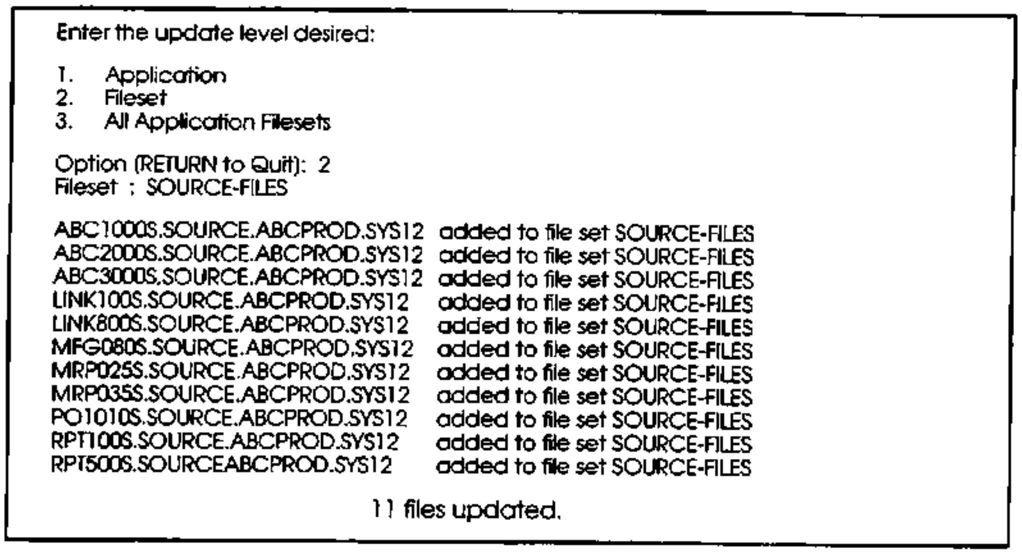
Auto Filesets (AF) Screen Fields

This section describes the fields on the AF screen for defining the files that belong to a fileset.

Master Fileset	The name of a fileset as defined on the Filesets (FS) screen.
INclude/EXclude	Indicates whether files identified by the descriptor should be included or excluded from the fileset.
System	The system where the files are located. If no system is specified, LIBRARIAN defaults to the current system.
Auto Fileset Descriptor	The general location of files associated with the fileset defined in the Master Fileset field.

Using AUTOUPDATE for New Master Files

You can run the Auto Fileset Update (**AUTOUPDP**) program at any time to introduce new files as masters. This program uses wildcard fileset descriptors entered through the Auto Filesets (AF) or saved on the Files in Filesets (FF) screen (described below) to locate the appropriate files on disk and load their filenames into the database to begin tracking them as master files. Run the program by typing **AUTOUPDATE**. Alternatively, you can type AUTO in the upper left box on the AF screen to run **AUTOUPDATE**. Figure 3–5 shows the **AUTOUPDATE** command for the SOURCE-FILES fileset.



```
Enter the update level desired:

1.   Application
2.   Fileset
3.   All Application Filesets

Option (RETURN to Quit): 2
Fileset : SOURCE-FILES

ABC1000S.SOURCE.ABCPROD.SYS12  added to file set SOURCE-FILES
ABC2000S.SOURCE.ABCPROD.SYS12  added to file set SOURCE-FILES
ABC3000S.SOURCE.ABCPROD.SYS12  added to file set SOURCE-FILES
LINK100S.SOURCE.ABCPROD.SYS12  added to file set SOURCE-FILES
LINK800S.SOURCE.ABCPROD.SYS12  added to file set SOURCE-FILES
MFG080S.SOURCE.ABCPROD.SYS12   added to file set SOURCE-FILES
MRP025S.SOURCE.ABCPROD.SYS12   added to file set SOURCE-FILES
MRP035S.SOURCE.ABCPROD.SYS12   added to file set SOURCE-FILES
PO1010S.SOURCE.ABCPROD.SYS12   added to file set SOURCE-FILES
RPT100S.SOURCE.ABCPROD.SYS12   added to file set SOURCE-FILES
RPT500S.SOURCEABCPROD.SYS12    added to file set SOURCE-FILES

                    11 files updated.
```

Figure 3-5. AUTOUPDATE Command

Using the Files in Filesets (FF) Screen

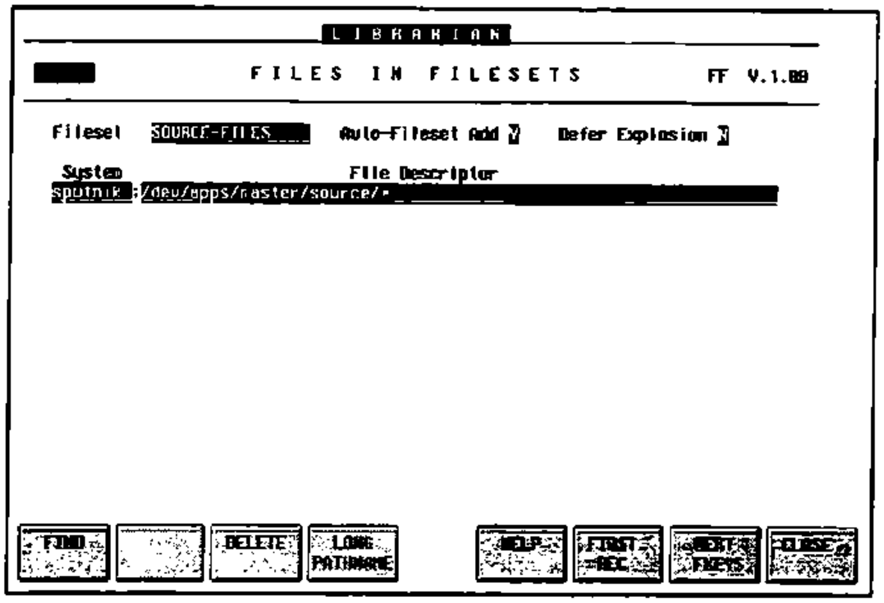
The Files in Filesets (FF) screen provides an alternate method of adding master files to library filesets. This screen is recommended when you need to add specific individual files to filesets without using wildcards. You can also use the FF screen to delete files from filesets.

The Files in Filesets screen combines most of the features of the AF screen with an immediate auto update. It is only practical when adding small numbers of files, since you must wait for filenames to be added to the database prior to proceeding to the next fileset descriptor.

When you press ENTER, LIBRARIAN explodes the descriptor to load the database with information for all existing files that match the descriptor. The files are listed on the screen as they are processed. If you do not want the files to be listed on the screen, use the SHOW FILES function key (set 2) to turn the file display *off*.

To load file information in batch or all at once, set both Auto Fileset Add and Defer Explosion to Y. Descriptors are loaded in the database, but files are not added to filesets until you run the **AUTOUPDP** program by using **AUTOUPDATE.**

Figure 3-6 shows the use of the FF screen where files are added to the sample SOURCE-FILES fileset.



```
                         L I B R A R I A N
              F I L E S   I N   F I L E S E T S           FF  V.1.00

  Fileset  SOURCE-FILES____  Auto-Fileset Add Y   Defer Explosion N

   System                   File Descriptor
  sputnik  :/dev/apps/master/source/*
```

Figure 3–6.   Files in Filesets (FF) Screen

Files in Filesets (FF) Screen Fields

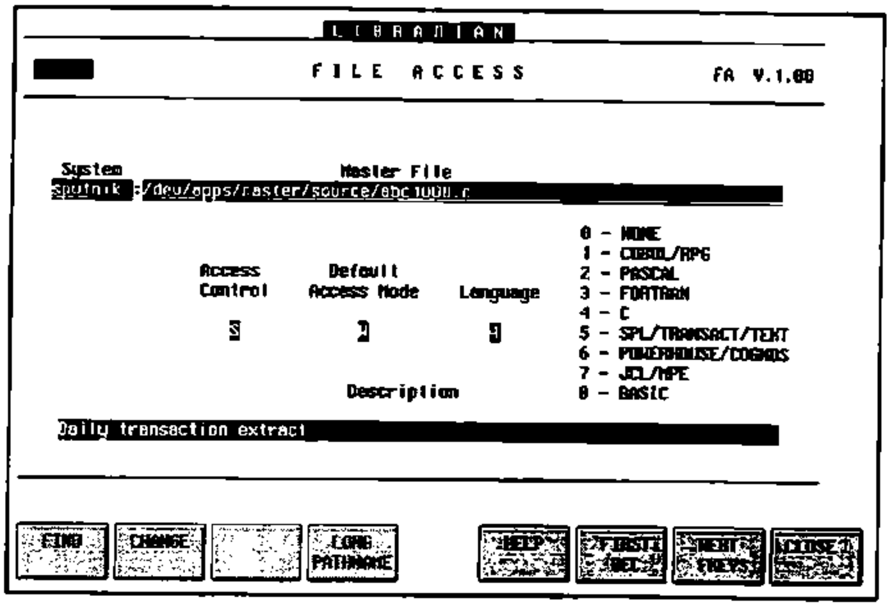
This section describes the fields on the FF screen for adding files to a fileset.

Fileset	The logical name for a set of files, as defined on the Filesets (FS) screen.
Auto Fileset Add	Indicates if the fileset descriptor will be added as an auto fileset descriptor (see AF screen) for later automatic fileset updates. Default: N. To add descriptor as an AF record, set to Y.
Defer Explosion	Indicates when to perform the explosion for this fileset. Default: N. To defer explosion until the **AUTOUPDP** program runs, set to Y. Make sure the Auto Fileset Add field is also set to Y.
System	System where the file is located. If no system is specified, LIBRARIAN defaults to the current system.
File Descriptor	The file descriptor, identifying the location of files in this fileset.

Customizing File Access Rules

When you assign files to filesets, LIBRARIAN assigns access rules to each file based on the default access parameters for the fileset. Use the File Access (FA) screen to review or customize the access parameters for individual master files. For example, if you assign serial access control to the fileset, but want some files in that fileset to have exclusive access, use the FA screen to change the access control for those files. You can add a one line text description for each file, as well.

Figure 3–7 shows the FA screen containing file access information.



```
                              L I B R A R I A N
                           F I L E   A C C E S S                    FA V.1.00

 System                     Master File
 sputnik :/dev/apps/master/source/abc1000.c
                                                      0 - NONE
                                                      1 - COBOL/RPG
            Access     Default                        2 - PASCAL
            Control    Access Mode       Language     3 - FORTRAN
                                                      4 - C
              S            2                 5        5 - SPL/TRANSACT/TEXT
                                                      6 - POWERHOUSE/COGNOS
                                                      7 - JCL/MPE
                          Description                 8 - BASIC
 Daily transaction extract
```

Figure 3-7. File Access (FA) Screen

File Access (FA) Screen Fields

This section describes the fields on the FA screen for reviewing and tuning the access parameters for specific files.

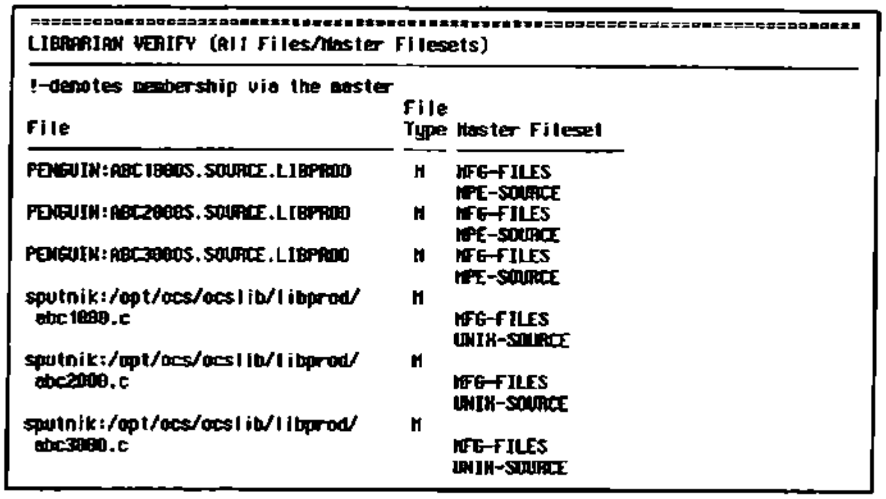
System	The system where the file is located. If no system is specified, LIBRARIAN defaults to current system.
Master File	The name of a file that is part of the library.
Access Control	The access control level for the master file (eXclusive, Read, Write or Multi–write).
Default Access Mode	The default access mode when copying files if it is not specified when performing a step.
Language	Controls the commenting style used in source annotation.
Description	A description of the file.

Reviewing Library and File Information

You can review detailed information about any group of files with the **VERIFY** function in LIBRARIAN. Some possible uses of this command include displaying the:

- associated Master file (or Delta file) - Format 3
- associated Master fileset(s) - Format 4
- location of Write mode copy - Format 9

Figure 3–8 shows the master fileset information displayed by **VERIFY** format 4.



```
=========================================================================
LIBRARIAN VERIFY (All Files/Master Filesets)
_________________________________________________________________________
!-denotes membership via the master
                                      File
File                                  Type Master Fileset
_____________________________________ ____ _______________
PENGUIN:ABC1000S.SOURCE.LIBPROD        M   MFG-FILES
                                           MPE-SOURCE
PENGUIN:ABC2000S.SOURCE.LIBPROD        M   MFG-FILES
                                           MPE-SOURCE
PENGUIN:ABC3000S.SOURCE.LIBPROD        M   MFG-FILES
                                           MPE-SOURCE
sputnik:/opt/ocs/ocslib/libprod/       M
 abc1000.c                                 MFG-FILES
                                           UNIX-SOURCE
sputnik:/opt/ocs/ocslib/libprod/       M
 abc2000.c                                 MFG-FILES
                                           UNIX-SOURCE
sputnik:/opt/ocs/ocslib/libprod/       M
 abc3000.c                                 MFG-FILES
                                           UNIX-SOURCE
```

Figure 3–8. Associated Master Filesets

You can also review detailed historical data for a specific file through the File Inquiry (FI) screen. Refer to Chapter 5, "Screens" in the *LIBRARIAN/iX Reference Guide.*

Table 3–1 lists LIBRARIAN reports providing important information on library structure and fileset membership.

Table 3–1. Library Information in Standard Reports

Report Code	Title	Description
RFE10/20	Fileset Explosion	Shows application hierarchy and physical files.
RFD10	Fileset Status	Shows status of master and secondary files in each fileset.
RFD20	Master File Status	Shows status of master files in each fileset.
RAF10	Auto Fileset Explosion	Shows hierarchy and fileset descriptors for each fileset.

Deleting Library and File Information

Break the relationship between two filesets by using **FIND** and **DELETE** on the Fileset Components (FC) screen.

Break the relationship of one or more files within a fileset by using **FIND** and **DELETE** on the Files in Filesets (FF) screen. Delete files collectively by using wildcards and pressing **DELETE** twice.

Delete a fileset by using **FIND** and **DELETE** on the Filesets (FS) screen. You can also use **DELETE** on this screen to delete all of the filesets components and member files. Mass deletion is not recommended if you have files or filesets belonging to more than one application.

Delete the data for an entire application's files and file movements by using **FIND** and **DELETE** on the Applications (AP) screen, or use the **DELETE** command.

Delete database information for files that no longer exist on disk by running the File Exception Report (RFX10) to identify nonexistent files, then use the **CLEANDB** command to delete the data for those files.

Summary

Table 3–2 summarizes the sequence for defining the master library for an application. Repeat this sequence for each application. Page references indicate the location where each activity is discussed in this chapter.

Table 3–2. Data Entry for Master Libraries

Activity	Screen	Code
1. Define an Application (Figure 3–1)	Applications	AP
2. Define Filesets (Figure 3–2)	Filesets	FS
3. Define Fileset Hierarchy (Figure 3–3)	Fileset Components	FC
4. Identify Physical Members of Filesets (Figures 3–4, 3–6)	Auto Filesets, Files in Filesets	AF FF
5. Load Database with Master Filenames (Figures 3–5)	AutoUpdate	AUTO
6. *Customize Data for Specific Files (Figures 3–7)	File Access	FA

* Asterisk indicate optional activities

Note Be sure to complete the data entry for an application library before proceeding to Chapter 4, "File Movement Rules".

File Movement Rules 4

This chapter describes how to use the screens to define or tune rules for users to obtain copies, replace master files with revisions, and perform other file operations. The following topics are discussed in this chapter:

- Steps and Routes
- Defining a Route
- Defining Basic Step Data
- Defining Options for Steps
- Example of a Step Definition
- Defining Rules for Introducing New Files
- Defining Step Refinements and Exceptions
- Reviewing File Movement Rules
- Summary

Steps and Routes

File movement rules including dependencies for an application are defined in LIBRARIAN as *routes* and *steps*.

A step defines the movement of files from one location to another. The step is the basic unit of the file movement system. Steps are grouped into routes. Steps are executed in LIBRARIAN as commands.

A route reflects a cycle of specific file movements and checkpoints. A route is made up of a series of steps that move/copy files or record an event. The steps must occur in a particular order.

Routes can include:

- maintenance for a current release
- development of a new release
- distribution procedure for updating data and/or program files
- control of programs for demonstration or test purposes

Steps can include:

- copying files from the library to a programmer's work group for modification
- copying new vendor files to all systems in a network
- approving modified files
- moving modified files from a programmer's work group to the QA account for testing
- moving tested and approved source and object code into the library
- introducing a new file into an application

Defining a Route

You should determine the routes and steps you want to define for an application. We recommend that you create flow diagrams to show how files are moved and copied from one location to another.

Figure 4–1 is a flow diagram for a basic development route consisting of three steps. In this route, programmers use the MFG–OUT step to obtain write mode copies of files from the master library. After programmers incorporate their changes, the manager uses the MFG–APPROVE step to approve the modified secondary files. Then, operators use the MFG–IN step to move the approved files back to the library.

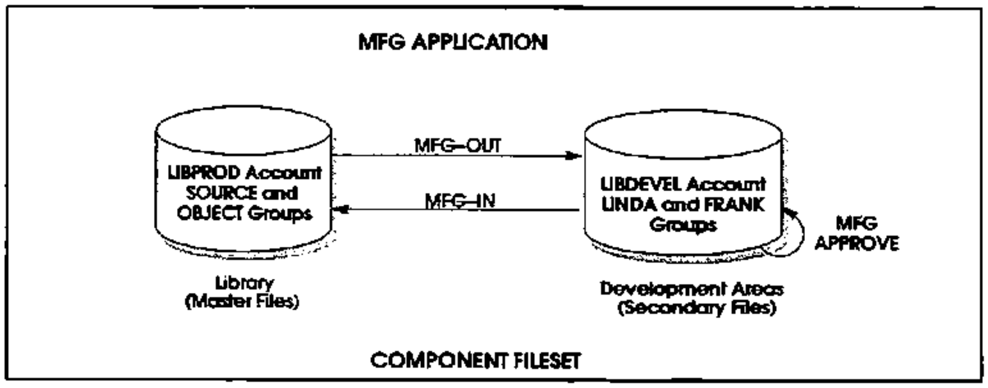
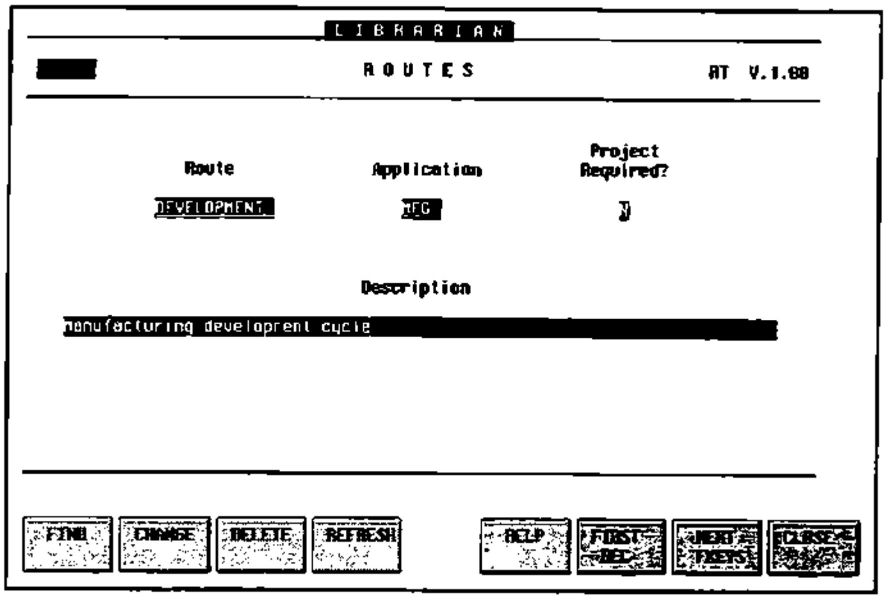

MFG APPLICATION
LIBPROD Account SOURCE and OBJECT Groups
MFG–OUT
MFG–IN
LIBDEVEL Account LINDA and FRANK Groups
MFG APPROVE
Library (Master Files)
Development Areas (Secondary Files)
COMPONENT FILESET

Figure 4–1. Sample Route Flow Diagram

First, assign a route name to describe a file movement cycle for the application. Use the Routes (RT) screen to assign a route name up to twelve characters. You can also add a description. Each route belongs to a specific application.

Note To require that all file movements within this route be linked to projects, set the Project Required field to Y. For more information about projects, refer to Chapter 6, "Projects".

Figure 4–2 shows the definition of the DEVELOPMENT route in the MFG application. This sample route does not require projects to be specified.


L I B R A R I A N

R O U T E S RT V.1.00

Route: DEVELOPMENT
Application: MFG
Project Required?: N

Description
manufacturing development cycle

Figure 4–2. Routes (RT) Screen

Defining Basic Step Data

Step definitions are the key to your LIBRARIAN implementation. A route can have as many steps as necessary to complete a cycle. The step definition identifies the type of operation, the files to which the step applies, and step options that control its behavior.

You perform a step by using the step name as a command. In menu mode, you can select the desired step from a menu by selecting the **Steps** option from the **File** menu.

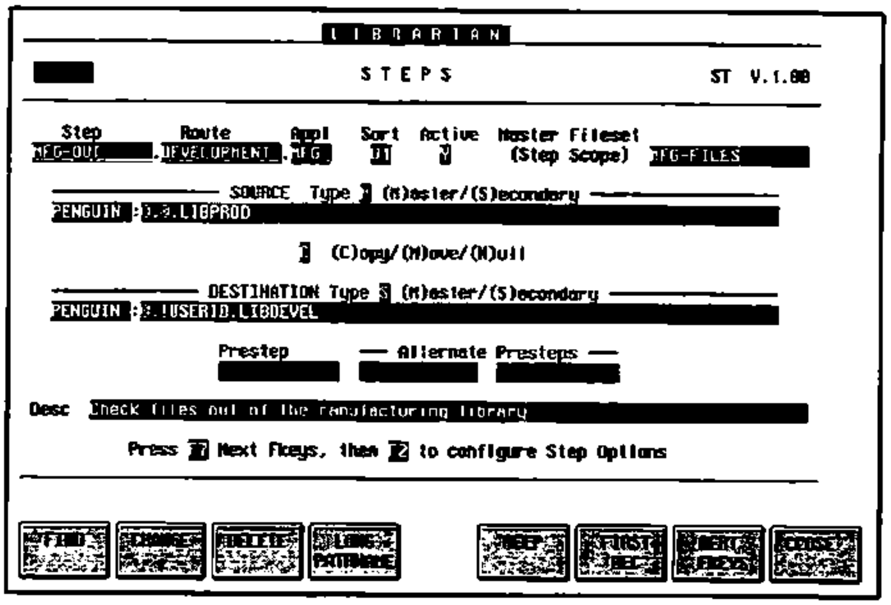
If you type a step name at the LIBRARIAN prompt, LIBRARIAN performs the named step, using the defined step parameters. For more information see the **PERFORM** command in Chapter 1, "Commands", in the *LIBRARIAN/iX Reference Guide.*

Step definitions can be specific or general.

- If step definitions are very general, you need to specify more information when performing the step in LIBRARIAN.
- If step definitions completely define all allowed files and options, you do not need to specify much detail when performing the step in LIBRARIAN.

Note When a user requests a step that exists in more than one route and/or application, an "ambiguous step name" message is issued, and a menu of steps is displayed. This menu of steps is alphabetically sorted and only displays steps the user is authorized to use.

Use the Steps (ST) screen to define steps (Figure 4–3). There are many options available when defining steps. Basic step information is described in this section, followed by advanced options in the next section.



```
                              L I B R A R I A N

                                  S T E P S                          ST  V.1.00

 Step          Route        Appl    Sort   Active   Master Fileset
MFG-OUT     .DEVELOPMENT  .MFG      01      Y        (Step Scope)   MFG-FILES

  ------------------ SOURCE  Type M (M)aster/(S)econdary ------------------
  PENGUIN :@.@.LIBPROD

                            C  (C)opy/(M)ove/(N)ull

  ------------------ DESTINATION Type S (M)aster/(S)econdary ----------------
  PENGUIN :@.!USERID.LIBDEVEL

                      Prestep       -- Alternate Presteps --

Desc  Check files out of the manufacturing library

          Press f8 Next Fkeys, then f2 to configure Step Options
```

Figure 4-3. Steps (ST) Screen

When a user requests a step that exists in more than one route and/or application, an "ambiguous step name" message is issued, and a menu of steps is displayed. This menu of steps is alphabetically sorted and only displays steps the user is authorized to use.

Step Identification

Information for identifying steps includes the following fields on the Steps (ST) screen:

Step.Route.Appl	This unique combination of step, route, and application identifies a step. If the step name is unique, you can perform the step without specifying the route and application. To minimize effort, do not use the same step name in more than one route or application. For example, instead of using a CHECKOUT step in several applications, you could define AP-OUT, GL-OUT, PAY-OUT, etc. as unique stepnames.
Sort	A unique number (1 to 99) for each step within a route, used for sorting. This number determines the order in which steps are listed on reports. The step number does not enforce sequences for performing steps. Presteps, discussed below, are used to enforce sequence.
Active	Indicates whether the step is currently active. The default is Y, indicating the step is active. To deactivate a step without deleting it, set this field to N.
Master Fileset	Limits the scope of the step to the specified fileset, restricting the files on which the step can operate. The scope can be the application fileset or any of its subsets.

- If you specify the application fileset, the step applies to any files that are part of the application.
- If you specify a lower-level fileset, the step is restricted to that specific subset of files.

For example, you can limit a step to the fileset SOURCE-FILES, which is a component of MFG-FILES. Members of MFG-FILES that are not members of SOURCE-FILES are not within the scope of the step and cannot be processed with this step.

The source location, discussed later in this chapter, can be used to further restrict the files on which the step can operate.

Step Operation

Each step performs one of the following three types of operations:

Copy	Copies files to another location.
Move	Physically moves files to another location, so they no longer exist in the original location.
Null	Does not perform any file movement, but records some event such as an approval.

Use a *copy* operation to obtain work copies of files for modification. Use a *null* operation to approve changes. Use a *copy* or *move* operation to send modified files to QA.

Each step defines a source location and destination location. You specify whether the files in each location are master or secondary files. This tells LIBRARIAN which file types are valid when authorizing files for the step, and is reflected in the *step type* which appears in reports. You can use system variables to define source and destination locations for steps.

Table 4–1 lists the step types, the typical file movement functions, and examples of stepnames for each type.

Table 4-1. Step Types

Step Type	Description	Function	Stepname Examples
MS	Master-to-Secondary	Copy	OUT, RELEASE, CHECKOUT, GET
SS	Secondary-to-Secondary	Move	TEST, QA, SUBMIT
		Copy	NEW, APPROVE, OK
		Null	TESTED
SM	Secondary-to-Master	Move	IN, PUT, CHECKIN
		Copy	REVISE, UPDATE

Master-to-secondary steps are *checkout* steps. Secondary-to-master steps are *checkin* steps. Secondary–to–secondary steps are all the steps in between.

Step operations are restricted by step type. For example, master-to-secondary steps can copy files, but cannot move them to another location. Null operations can only be performed on secondary-to-secondary steps. On secondary-to-master steps, the secondary file must have write mode access.

Source and Destination Locations

Source Location

The source location indicates where to locate valid files for the step when users specify relative pathnames (unqualified filenames). Use wildcards as needed to describe valid files for the step. When you execute a step, LIBRARIAN only authorizes files associated with the master fileset for the steps that are within the scope of the source location.

Note When you execute a step, the files you specify are relative to the step definition and not your current working directory.

Destination Location

Destination locations can be as specific or general as you need. It is a good practice to make the destination as specific as possible so that users do not have to specify a destination when executing the step. Use wildcards and edit masks, as described below, to allow LIBRARIAN to automatically determine destinations when users perform steps.

Equal (=) vs. At (@)

You can use the equal (=) sign in any element of a destination location. This indicates that files created by the step must have the same value as the corresponding element of its associated master file. For example:

MPE
```
Source Location:      SYSA:@.@.DEVEL
Destination Location: SYSA:=.=.QA
```
UNIX
```
Source Location:      sysa:/devel/appl/*/*
Destination Location: sysa:/qa/appl/=/=
```

The at (@) sign in MPE (asterisk (*) in UNIX) carries the element forward from the source filename to the destination filename, by default. The user can override this.

Wildcards

The equal (=), at (@), asterisk (*) , question mark (?) , and minus (–) signs can be used in combination with literal characters in the destination to edit or transform the corresponding source filename. See the section on Edit Masks at the beginning of Chapter 1 in the *LIBRARIAN/iX Reference Guide.*

The following example would copy files to a destination name that has a T appended to it:

MPE
```
Destination location: SYSB:@T
```
UNIX
```
Destination location: sysb:*t
```

When checking a secondary file back into the library using secondary-to-master steps, LIBRARIAN always replaces the master that it came from. This is true even if the secondary file gets renamed at some point during the route.

You can use system variables to define source and destination locations for steps. In addition to the wildcards @, ?, and # for MPE and *, ?, and regular expressions for UNIX, LIBRARIAN offers special wildcards to use in source and destination filenames: !LOGON, !USERID, !OWNER, and !MSUSER. These wildcards restrict the source and/or destination files to a particular location. As with literal elements, the user cannot override these values. In fact, if the element is omitted, it is automatically filled in.

Table 4-2 Special Wildcards

Wildcard	Where Used	Description
!LOGON !LOGIN	Source Destination	Substitute the user's current login automatically in an element. For UNIX, the login user is substituted as appropriate. For MPE, the login group, account, and system values are substituted as appropriate.
!USERID	Source Destination	Substitute the current LIBRARIAN user name in an element. For example, if programmers have their own work groups for secondary files and their user names are the same as their working directory, you could define the destination directory with **!USERID** on a checkout step.
!OWNER	Destination	Substitute the source file owner. For example, if several programmers submit files to QA, !OWNER can be used in the destination name for a single REJECT step that returns files to each programmer's area (e.g., MPE destination: SYSA:@.!OWNER..DVL; UNIX destination : sysa:/devel/!OWNER/*).
!MSUSER	Destination	Substitute the name of the LIBRARIAN user who originally checked out the file. This wildcard is useful in sending files which fail QA back to the original programmer's area.
!hpvar	Destination	Any MPE system variable that is prefixed by a !, e.g., !HPGROUP. The value is determined when a user performs the step.

Edit Masks for UNIX Pathnames

To carry forward, edit, or replace an element that is at the same level in both the source and destination filenames, follow the rules described above.

Because UNIX pathnames can have varying numbers of path elements (directories), you can edit (or skip) components at varying levels in the source filename using the following construct:

/(x [– y]) [*edit–mask*]

where x and y represent the desired range of components from the source pathname. x and y are numbers from 1 to *, where * is the last directory element of the pathname. If you want a specific element, omit y which is optional.

The optional edit mask is applied to each element in the range (do not include the brackets).

For example, the mask /(1–2)/devel/!USERID/(4–*)/= applied to the filename /usr/usr2/master/screens/abc results in the filename /usr/usr2/devel/milind/screens/abc.

You can also use the following wildcards in place of x or y:

~	number of levels in home directory path
.	number of levels in the current working directory path
..	one less than the number of levels in the current working directory path

You can use curly braces, i.e., { x [– y] }, to indicate mapping from the master file name rather than the current secondary file name.

For example, consider the following step called *demo–test*:

- Source files are defined as secondary files:

  sputnik:/usr/usr2/demo/dev/level1/level2/*

- Destination files are defined as secondary files:

  sputnik:/usr/usr2/demo/test/{ 5 – * } / =

  The edit mask { 5 – * } is evaluated using the associated master file path.

- Given the following source files:

  sputnik:/usr/usr2/demo/src/dir1/dir2/dir3/*

- The destination files would be expanded to the following:

  sputnik:/usr/usr2/demo/test/dir1/dir2/dir3/=

Prerequisites

This section describes how to define dependencies for steps.

Prestep

A *prestep* is a step in the current route that must be performed successfully before the current step can be performed on a file.

For example, if APPROVAL is a prestep for CHECKIN, the APPROVAL prestep *must* be completed before the file can be moved back to the library by the CHECKIN step. If you perform CHECKIN for several files, but some of the specified files were not approved with the APPROVAL step, those files will not be moved with CHECKIN.

Specify a prerequisite step by entering its step name in the Prestep field on the Steps (ST) screen (refer to Figure 4–3).

Alternate Presteps

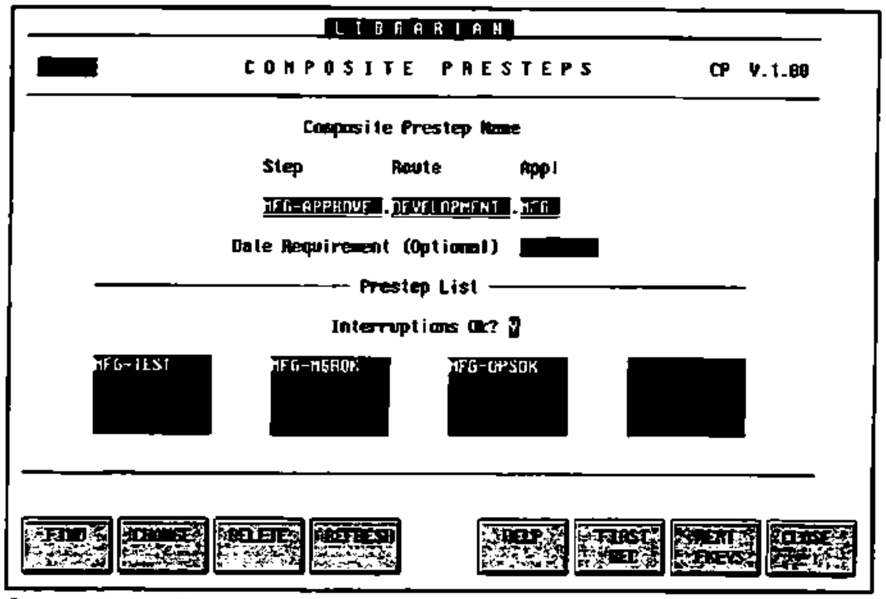
You can define a prestep and one or two *alternate* presteps. The prerequisite will be satisfied as soon as either the prestep or at least one of the alternate presteps is performed successfully.

For example, files can be copied to the development area from the master location (CHECKOUT step) or from QA (REJECT step). You can define CHECKOUT and REJECT as alternate presteps to the TEST step, which moves files to QA.

Multiple Prerequisites/Date Prerequisite

To require completion of more than one prestep, create a special *composite prestep* with the Composite Presteps (CP) screen as shown in Figure 4–4. A composite prestep is a list of previously defined steps that must be completed to satisfy the prestep. The order in which these steps are performed is not important. Use a composite prestep name in either the Prestep or the Alternate-Prestep fields of the Steps (ST) screen.

To prevent a step from being performed before a specific date, enter the date on the CP screen, then use the composite prestep name in any of the Prestep fields on the Steps (ST) screen.



```
                         L I B R A R I A N
        C O M P O S I T E   P R E S T E P S          CP V.1.00

                   Composite Prestep Name
              Step          Route          Appl
              MFG-APPROVE . DEVELOPMENT . MFG
           Date Requirement (Optional)
      ----------------- Prestep List -----------------
                   Interruptions Ok? Y
   MFG-TEST      MFG-MSROK      MFG-OPSOK
```

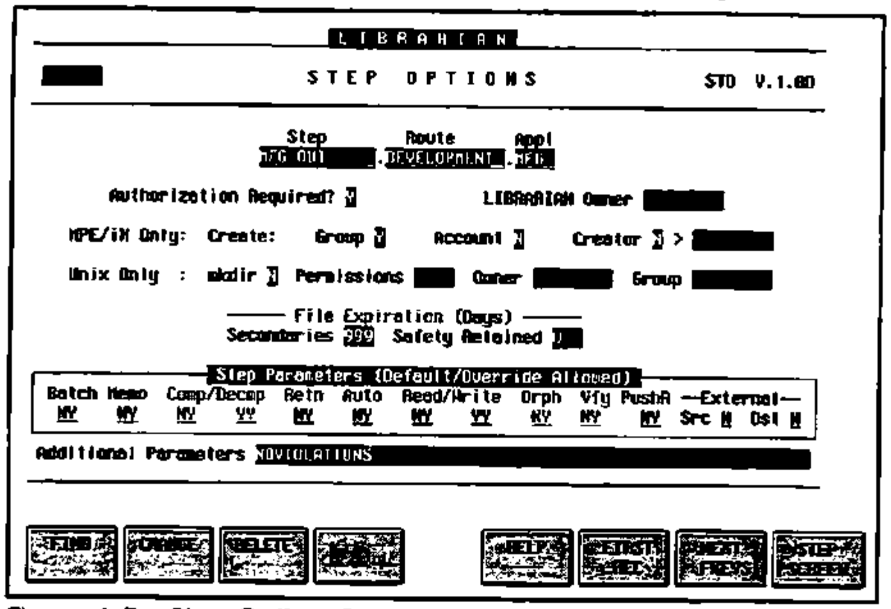
Figure 4-4. Composite Presteps (CP) Screen

Step Description

The description you enter for a step will appear on step reports as well as on the **Steps** menu that you access from the **File** menu.

Defining Options for Steps

You can define additional step options on the Step Options (STO) screen (see Figure 4–5) to customize the step to meet your needs. Initially, you can accept the defaults displayed and fine tune the step later.



```
                              L I B R A R I A N
                          S T E P   O P T I O N S                 STO  V.1.00

                          Step          Route       Appl
                         ?G OUT      .DEVELOPMENT_ .[illegible]

      Authorization Required? Y                LIBRARIAN Owner

   MPE/iX Only:  Create:    Group Y     Account Y     Creator Y >

   Unix Only   :  mkdir Y  Permissions      Owner           Group

                        ——— File Expiration (Days) ———
                        Secondaries 999  Safety Retained 1

              Step Parameters (Default/Override Allowed)
 Batch Memo  Comp/Decmp  Retn  Auto  Read/Write  Orph  Vfy  PushA  —External—
  NY    NY    NY    YY    NY    NY    NY    YY    NY    NY    NY   Src N  Dst N

 Additional Parameters NOVIOLATIONS
```

Figure 4–5. Step Options Screen

Authorization Required?	Indicates whether or not step authorization via the Step Authorizations (SA) screen is required for the step. Default: Y, indicates you must authorize specific users to perform the step. Setting this field to N allows all users to perform this step without authorization.
LIBRARIAN Owner	Sets the owner of secondary files LIBRARIAN creates during the step to the LIBRARIAN user you specify. Otherwise, the user performing the step becomes the owner.
Create: Group, Account, Creator	You can specify that the destination group, account, or creator should be created automatically by LIBRARIAN if it does not already exist by entering Y in the appropriate field.
⇒ Creator	You can specify the MPE file creator for files created by the step. If a file creator is not specified, LIBRARIAN uses the current MPE user for files created within the login account, and the default creator from the System Profile (SP) screen for all other files created.
mkdir	Instructs LIBRARIAN to create directories if they do not already exist.
Permissions	Sets the UNIX permissions on destination files.

Owner	Sets the UNIX owner for destination files.
Group	Sets the UNIX group for destination files.

Note

When a user requests a step that exists in more than one route and/or application, an "ambiguous step name" message is issued, and a menu of steps is displayed. This menu of steps is alphabetically sorted and only displays steps the user is authorized to use.

File Expiration

If you do not want read mode and retained copies to accumulate indefinitely, you can define an *expiration policy* (in days) for files created by the step. The expiration policy determines when the file is eligible to be flushed by the FLUSH utility.

You can define separate expiration periods for read mode secondaries and safety retained copies. Write mode copies and master files do not expire.

If you do not want files to expire or to be flushed, specify 999 days. The default value for *retained files* is 0 (indicating that the files expire the same day as created). The default value for *read mode secondaries* is 999 (indicating that the files do not expire).

Step Parameters (Defaults and Allowed Overrides)

For each step, you can specify a variety of default parameters for LIBRARIAN to use when you execute the step. You can also specify whether users can override these parameters when performing the step.

The list of parameters is located at the bottom of the Step Options (STO) screen. For each parameter, specify the default for the step (Y or N), and indicate whether users are allowed to override these defaults when performing the step (Y or N). Initially, all defaults are set to **N** to match the **PERFORM** command defaults, and all overrides are set to **Y**.

You can set the defaults for the following parameters in the step definition:

Batch	Authorizes the transaction online. Performs the actual file operation in a batch job.
Memo	Prompts for memo text describing the current transaction to be included with the log record.
Compress	Compresses the destination file.
Decompress	Decompresses the destination file (if compressed).

Retain	Retains existing destination files being replaced, if tracked.
Autoupdate	Adds files to appropriate filesets, on secondary-to-master steps, if any new files exist that match auto fileset descriptors in the database.
Read	Overrides default access mode and assigns read mode to the secondary.
Write	Overrides default access mode and assigns write mode to the secondary.
Orphan	For secondary files, breaks relationship to the master so it is no longer tracked by LIBRARIAN, if the destination is a secondary file. For master files, breaks its relationship with the application enabling you to assign a new application.
Verify	Causes files that have changed since they were created by LIBRARIAN to result in a violation.
Pushread	Allows a read mode secondary to replace its master file or a related write mode secondary. LIBRARIAN flags any write mode secondaries associated with the same master. Used for emergency fix steps.
External SRC/DST	Indicates that the source/destination files for this step are external to LIBRARIAN. Files are authorized, tracked and logged, but no files are created by LIBRARIAN, and no directory search is performed. Used to document creation of a file by an external process on remote computers, usually in a macro.

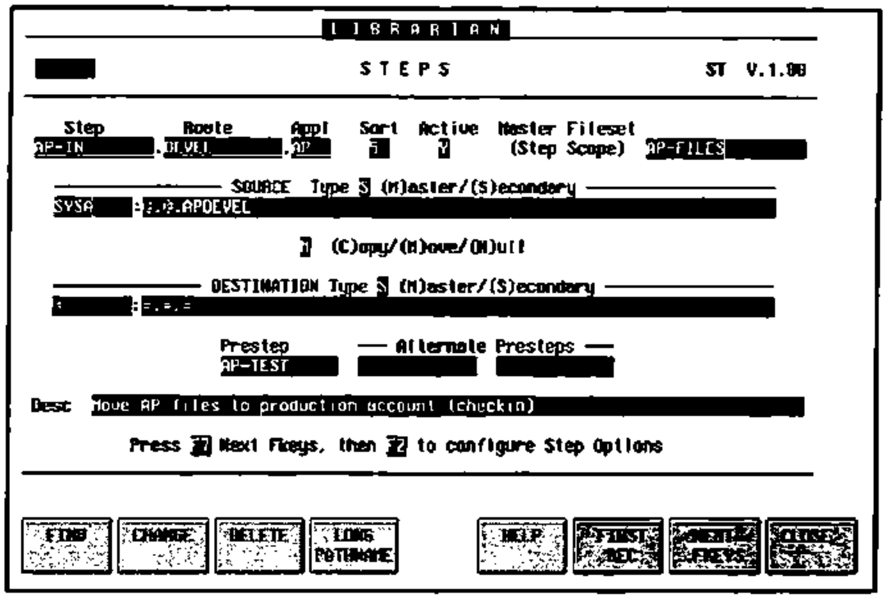
These examples depict how default parameters can be used:

- You can define a checkout step to automatically decompress files when the step creates files in a work area (**DECOMPRESS** default value is Y).
- You can require users to enter memo text when performing a step (**MEMO** default value is Y, and the override is N).
- You can define a checkin step so that replaced master files are always retained and the new masters are always compressed. (**RETAIN** default value is Y, and override is N; **COMPRESS** default value is Y and override is N).

Additional parameters can be hard–coded at the bottom of the STO screen. Refer to the **PERFORM** command in Chapter 1 of the *LIBRARIAN/iX Reference Guide* for information on valid parameters when performing a step.

Example of a Step Definition

Figure 4–6 and Figure 4–7 show a step definition for a secondary-to-master step called AP-IN.DEVEL.AP.



```
                         L I B R A R I A N
                            S T E P S                      ST  V.1.00

   Step          Route       Appl   Sort  Active  Master Fileset
 AP-IN        .DEVEL       .AP       5      Y      (Step Scope)  AP-FILES

            SOURCE  Type S (M)aster/(S)econdary
  SYSA     :@.@.APDEVEL

                       M  (C)opy/(M)ove/(N)ull

            DESTINATION Type S (M)aster/(S)econdary
  =        :=.=.=

              Prestep        Alternate Presteps
              AP-TEST

 Desc  Move AP files to production account (checkin)

       Press f8 Next Fkeys, then f2 to configure Step Options
```

Figure 4–6. Step Definition on ST Screen

The step is assigned number 5 for sorting purposes. The step is currently active and specific authorization is required to perform this step.

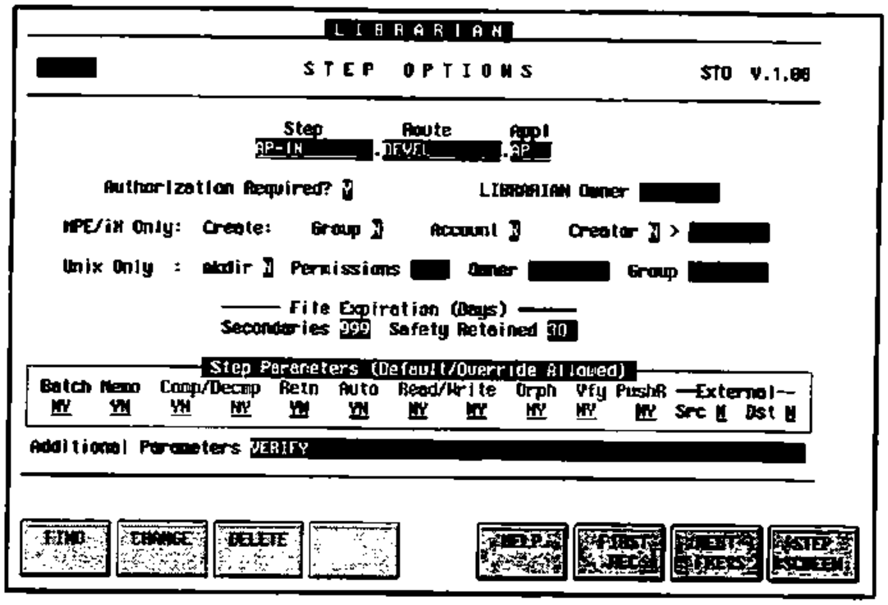
Because the master fileset for this step is AP-FILES, only secondaries associated with the AP-FILES fileset are processed by this step. Since the source file type is S (secondary), and the destination type is M (master), this step is identified on reports as type SM. This step moves files instead of copying them.

MPE

Because the source location is SYSA:@.@.APDEVEL, LIBRARIAN authorizes only secondaries of AP-FILES that are located in the APDEVEL account on SYSA. The destination location =.=.=.= shows that the files are returned to their original master locations.

The prestep AP-TEST must be performed before the AP-IN step.

Retained copies of master files expire in 30 days.



```
                              L I B R A R I A N
                         S T E P   O P T I O N S                  STO  V.1.08

                             Step        Route     Appl
                            AP-IN      . DEVEL    . AP

      Authorization Required? Y              LIBRARIAN Owner

   MPE/iX Only:  Create:    Group Y     Account Y     Creator Y >

   Unix Only  :  mkdir Y  Permissions      Owner          Group

                       ——— File Expiration (Days) ———
                       Secondaries 999  Safety Retained 30

                 Step Parameters (Default/Override Allowed)
 Batch Memo  Comp/Decmp  Retn  Auto  Read/Write  Orph  Vfy  PushR —External—
  NY    YN    YN    NY    YN    YN    NY    NY    NY    NY    NY   Src N  Dst N

Additional Parameters VERIFY
```

Figure 4-7. Step Options (STO) Screen

The following parameters are automatically invoked for this step:

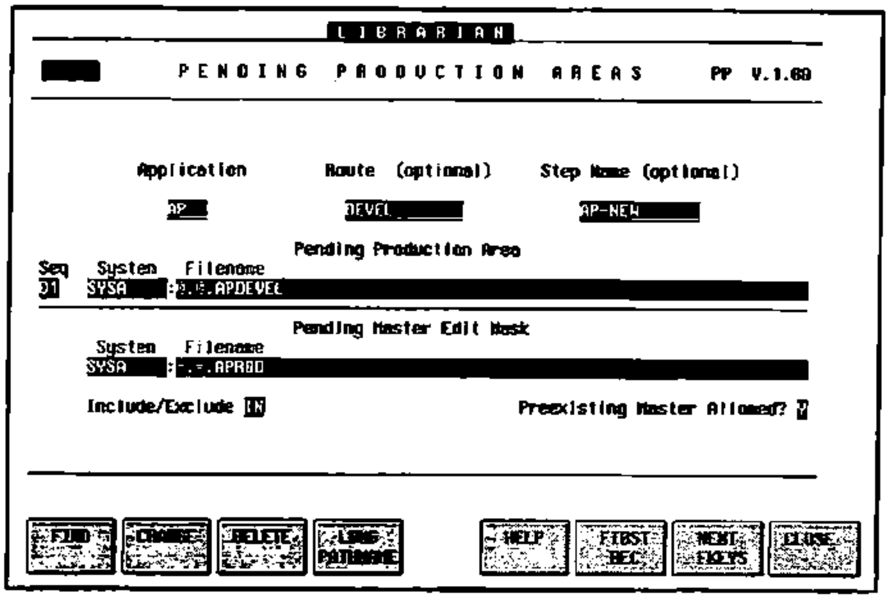
MEMO	Prompts the user for memo text.
COMPRESS	Compresses the new master files.
RETAIN	Retains the old master files.
AUTOUPDATE	Updates fileset membership automatically based on auto fileset descriptors, for new files or new filesets.
VERIFY	Does not allow files that have changed since they were moved to the test area.

The **BATCH**, **READ**, **WRITE**, **DECOMPRESS**, and **ORPHAN** parameters are not automatically invoked.

Defining Rules for Introducing New Files

Pending production areas are locations where you can introduce new files as secondaries during the development cycle. These locations are usually linked to specific steps. Pending production areas can be associated with any existing secondary step, or you can define a separate step to introduce new files. Use the Pending Production Areas (PP) screen to link a pending production area to a step (Figure 4–8).

The fields on the PP screen are used to specify one or more wildcard masks that limit or filter names of new files. Untracked files whose names are not in the scope of these wildcard masks do not qualify as new files and cause an "UNKNOWN FILES" violation. By using wildcards (=, ?, #,*, and @) in the Filename field, naming conventions can be enforced, and paths, groups, and/or accounts in which files can be introduced can be restricted.



```
                                L I B R A R I A N
            P E N D I N G   P R O D U C T I O N   A R E A S      PP V.1.60

          Application        Route  (optional)    Step Name (optional)

             AP              DEVEL                AP-NEW

                              Pending Production Area
Seq  System   Filename
01   SYSA    :@.@.APDEVEL

                              Pending Master Edit Mask
     System   Filename
     SYSA    :=.=.APROD

     Include/Exclude IN                       Preexisting Master Allowed? Y
```

Figure 4-8. Pending Production Areas (PP) Screen

Pending Master Edit Mask	Determines the name of the master file associated with each new secondary. Edit mask characters (=, @, ?,*, and, –) can be used to derive the master file name from the secondary filename (see *Edit Masks* at the beginning of Chapter 1 in the *LIBRARIAN/iX Reference Guide*). If the derived master filename does not already exist, LIBRARIAN creates a pending master record in the database to enforce serial write control and other authorization checks.
Preexisting Master Allowed?	Determines whether users can introduce new files when a master of the same name already exists. Usually, you do not want new source files to have the same name as masters that already exist. On the other hand, object code may or may not already exist in the library because these files are usually compiled in the development or test area without having been checked out.

To incorporate the introduction of new files created by programmers in development, create a step called AP-NEW by doing the following:

1. Use the Steps (ST) screen to define a null step called AP-NEW. Programmers perform AP-NEW to introduce new files in the development area. The source and destination locations for AP-NEW should be identical to the destination location for AP-OUT.
2. Use the Pending Production Areas (PP) screen to identify AP-NEW as a step where new files can be introduced, as shown in Figure 4–8. The new files must be within the fileset and source scope of the step.
3. Use the Steps (ST) screen to modify any subsequent step to have AP-NEW as an alternate prestep.

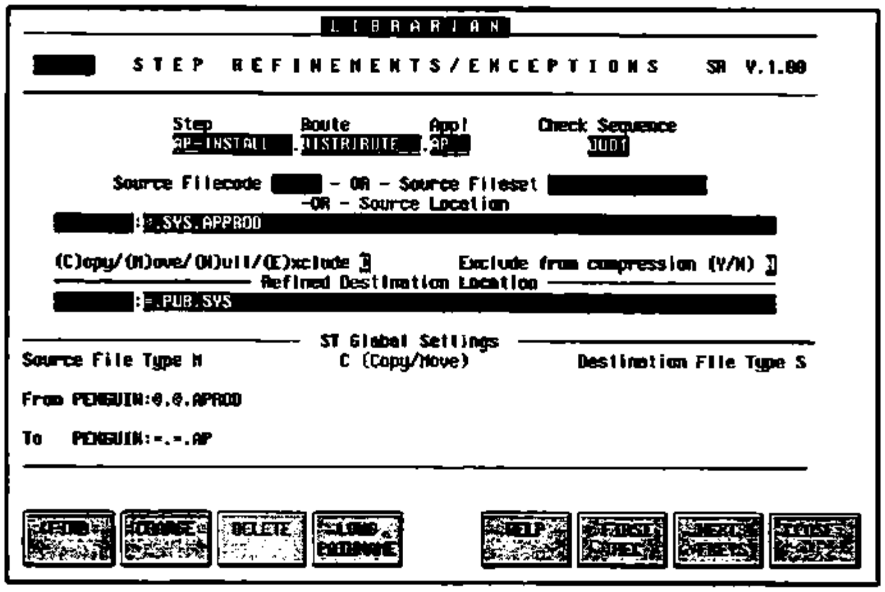
4. When a user performs the AP-NEW step on a file, it is identified as a pending production file. Untracked files that are not within the scope of the defined pending production area will be in violation of the rules (unknown files).

Defining Step Refinements and Exceptions

After defining a step, you can refine the way the step works for any subset of the source location using the Step Refinements/Exceptions (SR) screen. These refinements and exceptions, which can be based on name, filecode, or fileset membership criteria, include:

- **Different Operation** — An operation (move, copy, or null) different from the one defined for the step. For example, if a step is defined to move files, but there is a file that should be copied and not moved, type the filename in Source Location and enter code C.
- **Exclude Files** —Files which would otherwise be authorized for the step can be excluded by entering an E in the type field.
- **Different Destination** — A different destination for a subset of files. For example, if a step is defined to copy files to one area but you have a subset of files that should be copied to a different area, enter a mask in the Refined Destination Location field.
- **Multiple Refinements/Exceptions** — If you define several refinements or exceptions for a step (several different entries on the SR screen), the source locations could overlap. You must specify the sequence that LIBRARIAN checks for a matching refinement.

Figure 4–9 is a sample refinement/exception definition for the AP-IN step.



```
                              L I B R A R I A N
        S T E P   R E F I N E M E N T S / E X C E P T I O N S     SR  V.1.00

                  Step           Route          Appl       Check Sequence
                  AP-INSTALL    .DISTRIBUTE__  .AP__            0001

          Source Filecode      - OR - Source Fileset
                                 -OR - Source Location
                 :@.SYS.APPROD

   (C)opy/(M)ove/(N)ull/(E)xclude              Exclude from compression (Y/N)
   ---------------------- Refined Destination Location ----------------------
                 :=.PUB.SYS

   ------------------------- ST Global Settings ----------------------------
   Source File Type M         C (Copy/Move)            Destination File Type S

   From PENGUIN:@.@.APROD

   To   PENGUIN:=.=.AP
```

Figure 4–9. Step Refinements (SR) Screen

Notice that the check sequence is 0001, indicating this refinement is the first one LIBRARIAN checks when performing the step.

MPE The source location for the step is SYSA:@.@.APDEVEL. This refinement applies only to files in the SYS group. For this subset of files only, the destination location is the PUB group of the SYS account instead of the SYS group of the AP account. Note also that this exception copies the files instead of moving them.

Reviewing File Movement Rules

You can review the step definitions online with the **HELP STEPS** or **HELP *stepname*** commands.

LIBRARIAN reports provide important information on defined file movement rules, as shown in Table 4–3:

Table 4-3. File Movement Information on Standard Reports

Report Code	Title	Description
RAD10	Step Detail Report	Overview of all route and step definitions
RAD20	Step Detail Report (with related data)	Complete detailed information about all routes and steps

Sequence Summary

Table 4–4 summarizes the sequence for defining file movement rules for a route. Repeat this sequence for each route. Use figure numbers to locate where these activities are discussed. Perform activities 2 through 6 for each step before defining the next step.

Table 4-4. Data Entry for File Movement Rules

Activity	Data-Entry Screen	Code
1. Define route (Figure 4–1)	Routes	RT
*2. Define composite presteps (Figure 4–4)	Composite Presteps	CP
3. Define steps (Figure 4–6)	Steps	ST
*4 Define step refinements and exceptions (Figure 4–9)	Step Refinements/Exceptions	SR
*5. Define alternate search locations (Figure 7–2)	Forward Versioning	FV
*6. Define how new files can be introduced (Figure 4–8)	Pending Production Areas	PP

* Asterisks indicate optional activities

Users and Authorizations 5

Each user in LIBRARIAN is identified by a user ID and password. All LIBRARIAN IDs and passwords are case sensitive.

You can either assign special capabilities to users or you can authorize each user for the steps and projects they are allowed to work with.

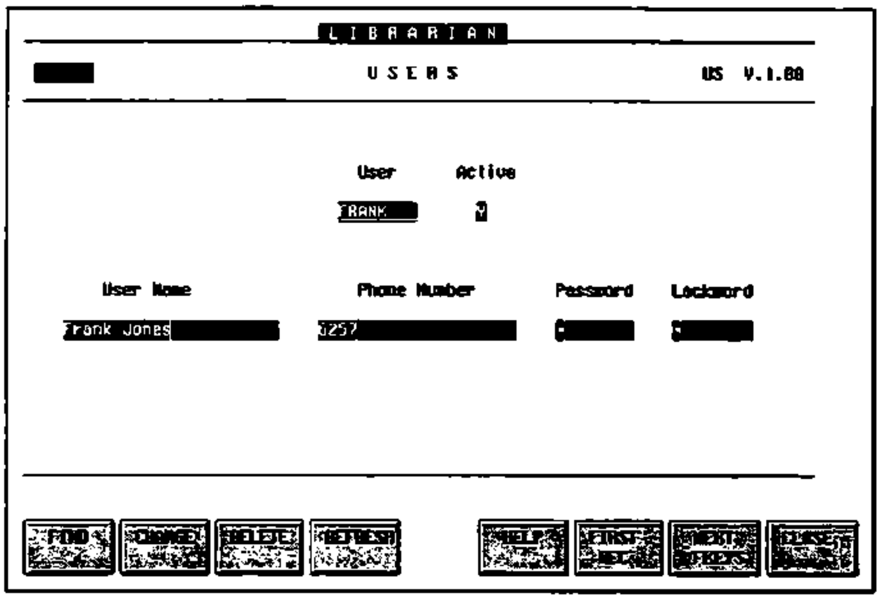
This chapter describes how to define user profiles and assign capabilities. The following topics are covered in this chapter:

- Defining Users
- Assigning User Capabilities
- Authorizing Users to Perform Steps
- Reviewing User Data
- Sequence Summary

Defining Users

To define users to LIBRARIAN, the LIBRARIAN Manager uses the Users (US) screen to assign user IDs. User IDs can be independent of system logons. Users can review and modify their own user information.

Figure 5–1 shows user information as defined on the Users (US) screen.



```
                         L I B R A R I A N
                             U S E R S                        US V.1.00

                          User      Active
                          FRANK       Y

      User Name           Phone Number       Password  Lockword
   Frank Jones            3257
```

Figure 5–1. Users (US) Screen

The user information includes the user's name, phone number, LIBRARIAN password, and user's lockword (the lockword is optional for MPE and displays only when you access your own record).

Note The User ID appears in the transaction log for each file movement. To have an audit trail that uniquely identifies individual users, assign a unique User ID to each user (i.e. do not allow users to share the same User ID).

The ACTIVE flag indicates whether the user record is currently active or inactive. Use the ACTIVE flag to suspend or reinstate a user's access to LIBRARIAN. The LIBRARIAN Manager can change this at any time. When set to "Y" (active), the user can access the system. When the flag is set to "N" (inactive), the user cannot access the system.

Users can access and maintain their own user information, but cannot access information for other users.

Each user can change their own personal password and lockword with the US screen, the **USER** command or the **User** menu. The user password protects against unauthorized use of LIBRARIAN. You must supply the correct password to access LIBRARIAN functions. Passwords and lockwords are encrypted in the LIBRARIAN database.

UNIX

If your UNIX login user matches your LIBRARIAN user, you will need not need to supply a password when you run LIBRARIAN.

Note It is not necessary to assign user passwords during the initial setup. The first time you use LIBRARIAN, you are prompted to assign a password for your User ID.

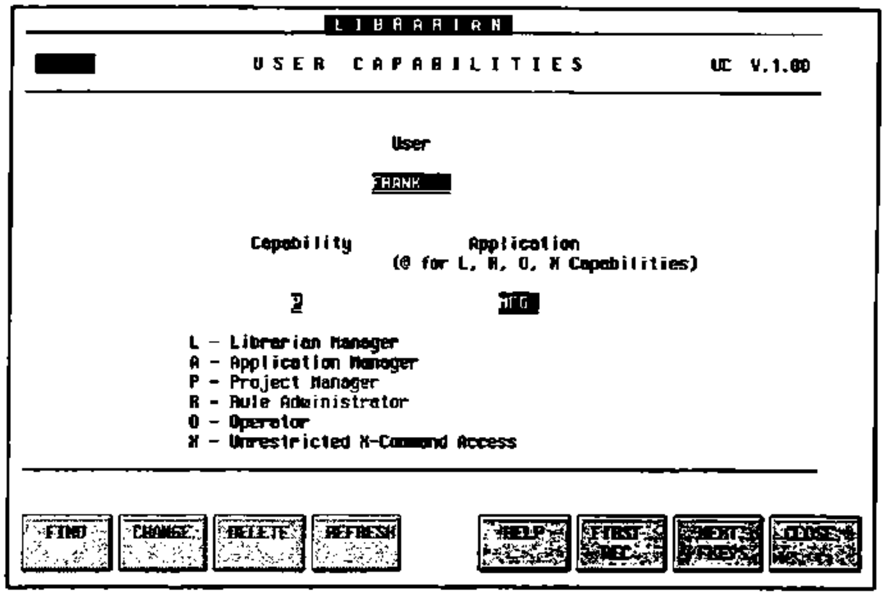
MPE

If a lockword is present, LIBRARIAN automatically assigns it to any files the user creates.

When you access this screen, the password and lockword fields display an asterisk(*), rather than their actual contents. You can display the contents using the SHOW PASSWORD function key. To hide the contents, press the HIDE PASSWORD function key. Using **F2 (Set 2)** toggles between the SHOW PASSWORD and HIDE PASSWORD functions. Users can only view their own passwords on this screen.

Assigning User Capabilities

The LIBRARIAN Manager can assign special capabilities to users through the User Capabilities (UC) screen. Figure 5–2 shows a sample user capability definition in the UC screen.



```
                         L I B R A R I A N
        U S E R   C A P A B I L I T I E S              UC V.1.00

                              User

                              FRANK

              Capability            Application
                              (@ for L, R, O, X Capabilities)

                  P                   MFG

            L - Librarian Manager
            A - Application Manager
            P - Project Manager
            R - Rule Administrator
            O - Operator
            X - Unrestricted X-Command Access
```

Figure 5–2. User Capabilities (UC) Screen

LIBRARIAN is provided with a predefined user name, LIBMGR, which has LIBRARIAN Manager capability. If you have not already done so, within LIBRARIAN create your own LIBRARIAN Manager on the US screen, and then delete the capabilities and user name for LIBMGR.

The following special capabilities can be granted to a user:

L (LIBRARIAN Manager)	Can perform all operations on all files on all systems. Defines and maintains the file management rules for the entire system.
A (Application Manager)	Can perform all operations on all files within assigned applications. Maintains file management rules for an application.
P (Project Manager)	Can define projects within an application and maintains project status. Authorizes users to work on projects (see Chapter 6, "Projects").
O (Operator)	Can use the **SHOWLOG>FLUSH** command to delete transaction log records and LIBRARIAN's **>RESTORE** command to restore retained files.
R (Rules Administrator)	Can define and modifies rules, and creates libraries. It cannot create users or perform file movement operations.

X (X Capability)	Can use the **X** commands in LIBRARIAN to operate on files unknown to LIBRARIAN, regardless of operating system security. Other users can use these commands but operating system security is enforced.

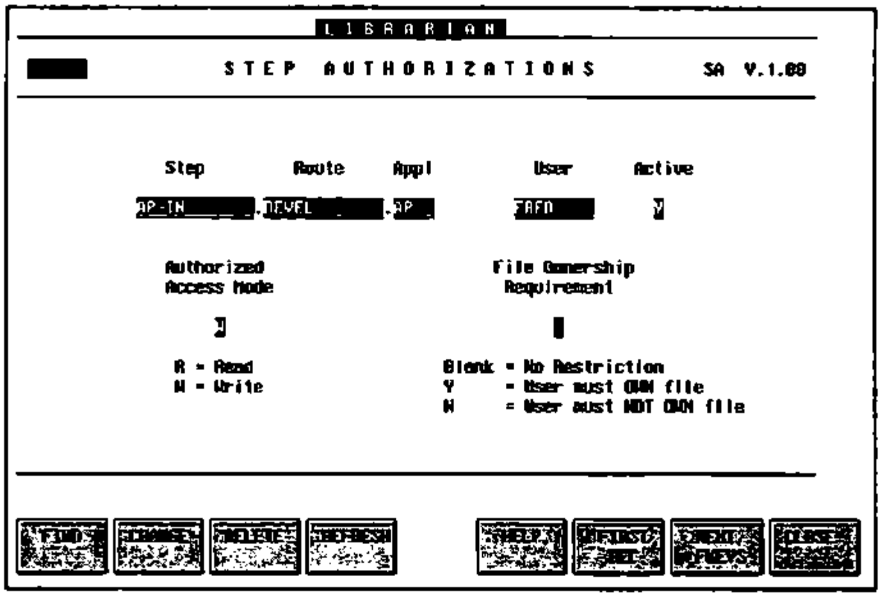
Authorizing Users for Defined Steps

If a step definition requires user authorization, LIBRARIAN checks for user authorization prior to performing the step.

- LIBRARIAN Managers can perform any step in any application.
- Application Managers can perform all steps within their own applications.
- General system users require explicit authorization.

You can authorize users for specific steps, routes, and/or applications. Any number of users can be authorized to perform a given step. Because each step, route, or application is authorized separately, users only need one user ID for a variety of operations.

Use the Step Authorizations (SA) screen to designate which users can perform each step. Figure 5–3 shows a typical step authorization for a single step as defined in the Step Authorizations (SA) screen.



```
                          L I B R A R I A N
                 S T E P   A U T H O R I Z A T I O N S            SA  V.1.00

             Step        Route     Appl         User      Active

            AP-IN       .DEVEL     .AP          FAFD        Y

             Authorized                      File Ownership
             Access Mode                      Requirement

                 R

              R = Read                  Blank = No Restriction
              W = Write                 Y     = User must OWN file
                                        N     = User must NOT OWN file
```

Figure 5–3.   Step Authorizations (SA) Screen

The following are descriptions for the fields on the SA screen:

Step Route Appl.	Name of the step, route and application being authorized. If you want to authorize a user to perform all steps in a route, use an at (@) sign in the Step field. If you want to authorize a user to perform all steps in the application, use an @ sign in the Step and Route fields.
User	Name of the user authorized to perform the step.
Authorized Access Mode	Specifies whether user is allowed to obtain write mode files with this step. For example, all programmers on a development team could be allowed to check out files in write mode because their modifications will eventually replace the current masters. Other users may need to obtain copies of files but should not be modifying them, so you would set their Authorized Access Mode to R (read–mode).
Active	Use this field to suspend or reinstate the step authorization for a user. This indicates whether the user is currently authorized to perform the step. Set this flag to "N" if you want to suspend the authorization without deleting the record from the database.
File Ownership	Use to further restrict the use of a step for files based on ownership. You can restrict users to files the user owns (e.g., a user can only submit his own files) or on files the user does not own (e.g., a user can only test files checked out and modified by a different programmer). The owner of a file is the user who created the file using LIBRARIAN.

Note

You can define a step that does not require authorization. Set the "Authorization Not Required" flag on the ST screen to Y. Steps that do not require authorization are available for use by any user, without specific authorization on the Steps Authorization (SA) screen.

Reviewing User Data

The following offline reports contain user data:

Table 5–1. User Data Reports

Report Code	Title	Description
RUD10	Users Report	Shows complete user profiles.
RUS10	Step Authorizations Report	Shows users who can perform each step.

Sequence Summary

Table 5–2 summarizes the sequence that has been described in this chapter for defining user information.

Table 5–2. Data Entry for User Authorizations

Activity	Data Entry Screen	Screen Code
1. Create user profile records. Optionally add passwords and lockwords. (Figure 5–1).	Users	US
2.* Assign special capabilities (Figure 5–2)	User Capabilities	UC
3. Authorize use of steps by specific users (Figure 5–3)	Step Authorizations	SA

* Asterisk indicates optional activities

Deleting or Inactivating Users

When a user leaves the company or changes jobs, you will want to delete that user and all of his/her authorizations from LIBRARIAN. However, since the user name may appear throughout the audit trail and file tracking information, it is generally a better practice to inactivate the user ID rather than deleting it. To inactivate, simply bring up the user ID on the US screen and change the Active flag to "N". If anyone attempts to use this ID, LIBRARIAN will respond as if the user ID did not exist.

When you are ready to completely delete a user ID, bring up the ID on the US screen and press the "Delete" function key (F3). You will be prompted to confirm; then the user ID and all related data, such as User Capabilities, Step Authorizations and Project Authorizations, will be deleted.

Projects 6

LIBRARIAN allows you to define projects to associate file movements with specific service requests, maintenance tasks, or software development projects. LIBRARIAN automatically tracks files that are worked on for a project, and identifies the programmer that changed them. Complete project information is available in standard reports and in customized SHOWLOG audit trail reports. You can restrict projects so that only authorized users are allowed to work on them. You can also require that all file movements be associated with a project.

This chapter describes how to set up and maintain projects with LIBRARIAN. The following topics are included:

- Defining Project Managers
- Creating Projects
- Authorizing Projects
- Changing Project Status
- Reviewing Projects
- Flushing Project Transaction Records
- Using Projects in LIBRARIAN
- Project Filesets
- Distributing Files by Project

Defining Project Managers

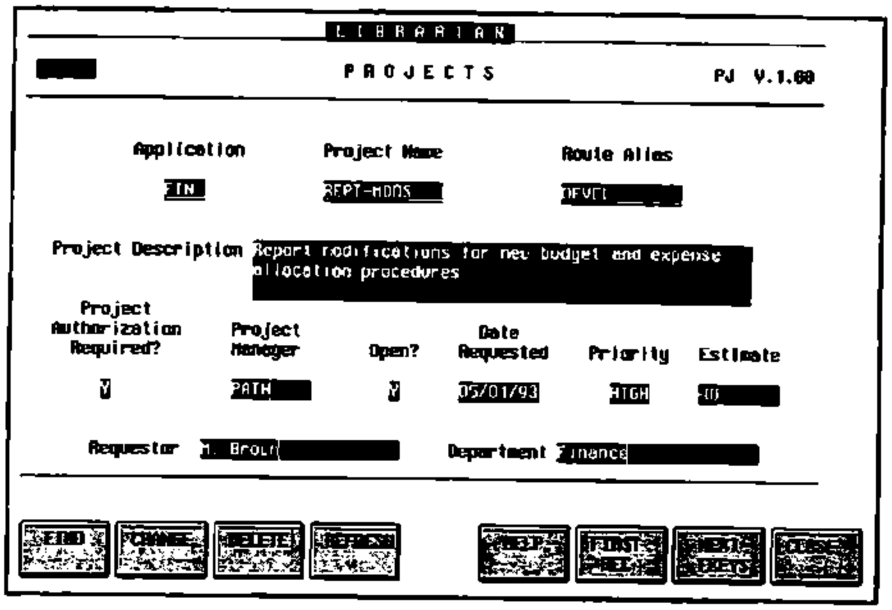
Application managers and the LIBRARIAN Manager can create and manage projects, and can assign Project Manager (P) capability to other users. Use the User Capabilities (UC) screen described in Chapter 5 to assign Project Manager capability.

Project Managers can create projects, maintain project status, and authorize users to work on projects. Project Managers are automatically authorized to work on any project they define.

Creating Projects

To create projects, you can use either the Projects (PJ) screen or the **PROJECT** command. This section covers the procedure for using the PJ screen. To use the **PROJECT** command, see Chapter 1, "Commands", in the *LIBRARIAN Reference Guide*.

Figure 6–1 is a sample project definition for the REPT-MODS project. This project is used for modifying budget and expense reports in the DEVEL route of the FIN application. Note that project authorization is required.



```
                         L I B R A R I A N
                          P R O J E C T S                    PJ V.1.00

      Application        Project Name          Route Alias

         FIN              REPT-MODS              DEVEL

  Project Description  Report modifications for new budget and expense
                       allocation procedures

    Project
  Authorization   Project                 Date
   Required?      Manager      Open?   Requested   Priority   Estimate

       Y           PATH          Y      05/01/93     HIGH       10

      Requestor  M. Brown                 Department  Finance
```

Figure 6–1. Projects (PJ) Screen

The PJ screen contains the following fields:

Application	Application to which the project belongs.
Project Name	Project name that users specify in LIBRARIAN transactions or select from a project menu. For each project, LIBRARIAN automatically creates a user fileset with the same name as the project. The specific files you work on for a project are automatically added to the project fileset on checkout steps and other steps that introduce new files. Files can also be added manually with FMAINT commands.
Route Alias	Associates a project with a route. The project name can be used as an alias for that route in LIBRARIAN operations. Projects can be defined for all routes in the application by specifying an at (@) sign in place of the route.
Project Description	A free form description of the project. This description appears on project menus and reports.

Project Authorization, Required?	A flag that determines whether users require specific authorization to work on a project.
Project Manager	Assigns a project to a specific Project Manager who can change project data and status information. The Project Manager can associate files with the project without specific authorization.
Open?	Allows you to define a project and reserve it for later use by setting the Open flag to "N". Set the flag to "Y" if you want the project to be available as soon as it is defined.
Date Requested, Priority, Estimate, Requestor, Department	Used for project documentation. It has no effect on LIBRARIAN file processing.

Defining Project Hierarchies

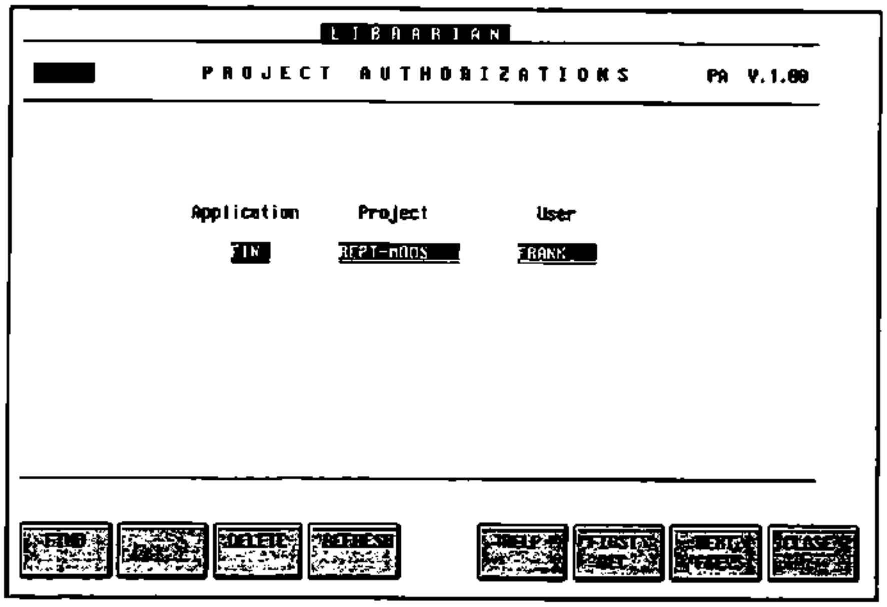
To create project hierarchies, use the **FMAINT RELATE** command to relate project filesets to other project filesets. As a result, when you check out files that belong to a project, these files automatically belong to any parent project fileset. You can then perform checkins, approvals and/or distribution by referring to parent project filesets.

Transactions are logged under the parent fileset. When you refer to them in commands, however, the last project for the file reflects the actual project name at the time of checkout.

Authorizing Projects

If the project requires project authorization, the Project Manager or Application Manager must authorize users to work on the project. Any number of users can be authorized to work on a project. Project Managers are automatically authorized to perform work for their own projects without requiring special authorization.

Use the Project Authorizations (PA) screen to authorize users to work on projects. Figure 6–2 is a sample project authorization for setting up a user, Frank, to work on the REPT-MODS project in the FIN application.

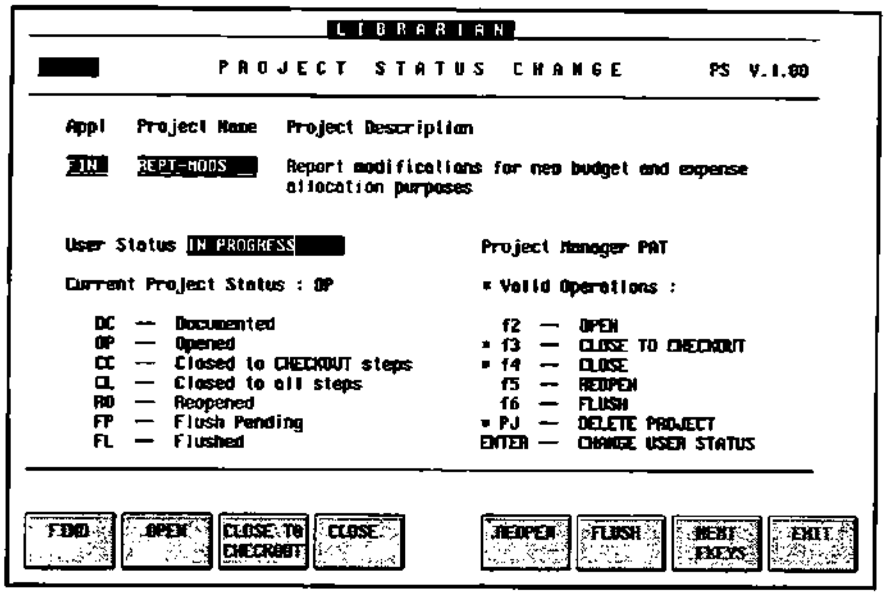


```
                              LIBRARIAN
            PROJECT  AUTHORIZATIONS                PA V.1.00

            Application    Project        User
               FIN         REPT-MODS      FRANK
```

Figure 6–2. Project Authorizations (PA) Screen

Changing Project Status

Use the Project Status Change (PS) screen to review and modify project status. Figure 6–3 shows a sample PS screen.



```
                              L I B R A R I A N
            P R O J E C T   S T A T U S   C H A N G E        PS V.1.00

   Appl   Project Name   Project Description

   FIN    REPT-MODS      Report modifications for new budget and expense
                         allocation purposes

   User Status IN PROGRESS                    Project Manager PAT

   Current Project Status : OP                * Valid Operations :

      DC  —  Documented                         f2  —  OPEN
      OP  —  Opened                           * f3  —  CLOSE TO CHECKOUT
      CC  —  Closed to CHECKOUT steps         * f4  —  CLOSE
      CL  —  Closed to all steps                f5  —  REOPEN
      RO  —  Reopened                           f6  —  FLUSH
      FP  —  Flush Pending                    * PJ  —  DELETE PROJECT
      FL  —  Flushed                          ENTER —  CHANGE USER STATUS
```

Figure 6–3. Project Status Change (PS) Screen

Possible project status values are:

DC	Documented (project was defined but not opened)
OP	Opened
CC	Closed to Checkout (project can be used, but not for master-to-secondary steps)
CL	Closed to All Steps
RO	Reopened
FP	Flush Pending
FL	Flushed

If you want to deactivate a project temporarily, use **F4** (CLOSE). To reactivate a closed project, use **F5** (REOPEN).

To flush the transaction log records for the project and project fileset, use **F6** (FLUSH). The project status will change to "Flush Pending". After flushing the project's log records with the FLUSHLOG utility, the project status changes to "Flushed". You can delete a project with a "Flushed" status on the Projects (PJ) screen, if you wish.

Reviewing Projects

Use the Project Inquiry (PI) screen to review all of the projects for an application.

You can select a specific project status, or one of the following status groups:

AL	All Projects
AI	All Inactive Projects
AC	All Closed Projects
AA	All Active Projects
AO	All Open Projects

Figure 6–4 shows a sample project inquiry for all projects in the FIN application. The same information is also available offline in the Project Detail Report (RPJ10).

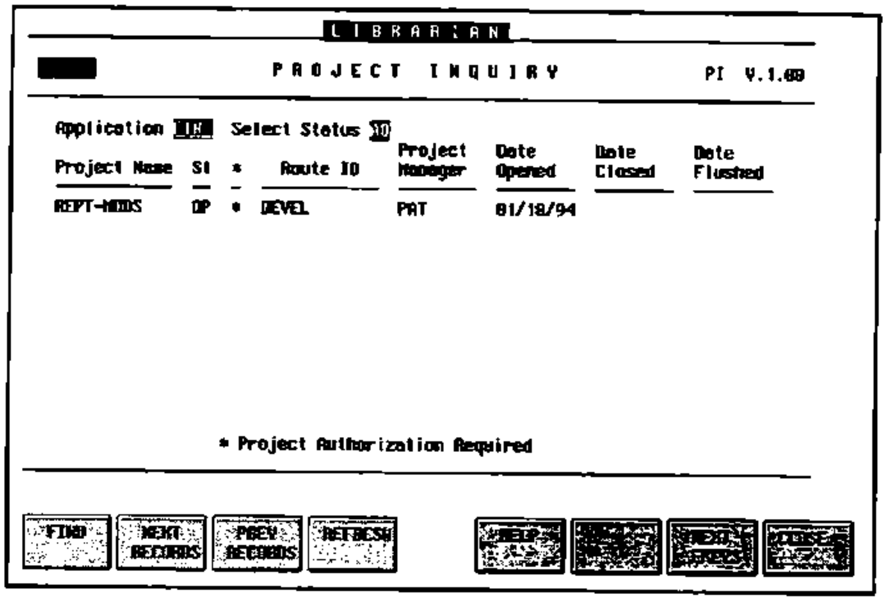

```
                              L I B R A R I A N
                      P R O J E C T   I N Q U I R Y              PI  V.1.00

Application FIN  Select Status AO
                                          Project   Date       Date      Date
Project Name  St  *   Route ID            Manager   Opened     Closed    Flushed
------------  --  -   --------            -------   --------   ------    -------
REPT-MODS     OP  *   DEVEL               PAT       01/18/94

                  * Project Authorization Required
```

Figure 6–4. Project Inquiry (PI) Screen

Flushing Project Transaction Records

Transaction records for projects are protected from being flushed, when you run the FLUSHLOG utility. Project audit trail records are saved until you change the status of a project to "Flush Pending" on the Project Status Change (PS) screen. The FLUSHLOG utility flushes log records for projects providing the project status is "Flush Pending". After running FLUSHLOG, the project status changes to "Flushed". Once flushed, you can delete the project record on the Projects (PJ) screen.

Using Projects in LIBRARIAN

When projects are defined for a route, you can associate files created by steps within that route with a specific project. If the route definition requires project identification, you must associate the work with a project. Once a secondary file has been associated with a project, it cannot be changed in a subsequent step — you must wait until the route has completed (i.e., the file checked in).

When you perform a step, you select a project from a menu of the open projects you are authorized to work on. If projects are optional for the route, you can select the "no project" option from the list.

Additionally, you can specify the project on the command line by substituting the project name for the route when performing a step. For example, to perform the AP-OUT step for the REPT-MODS project, type:

Project Filesets

Associated with every project is a fileset containing the master files that are checked out under that project. This is known as a *project fileset*.

Filenames are automatically added to the project fileset when files are checked out or new files are introduced for the project.

When you use either the **CLEANDB** or **PURGE** command to remove the last master, related secondary, or retained file, the master filename will automatically be removed from the project fileset.

Additionally, if you use either the **MOVE** or **RENAME** command to remove the last master, related secondary, or retained master associated with a project, the old filename will automatically be removed and the new one will be added.

Note Steps will automatically locate secondary file(s) in the step source location if you specify the project or project fileset.

Distributing Files by Project

You can imply the files associated with a project when performing a step by specifying the project name, rather than files. The syntax is:

```
>step.project
```

Alternatively, you can omit the project name and select your project from the project menu when projects are defined. In menu mode, this is the only alternative.

Subset selection by project selects only files associated with a particular project. This parameter must follow all file references, including destination locations, if specified. **PROJECT** is valid for all commands. The syntax is:

```
filelist; PROJECT=proj
```

If you use a step to copy files in read–mode (e.g., move–to–production), LIBRARIAN automatically copies the appropriate revisions of the files associated with the project that you specify. However, if you do not use a step for file distribution (e.g., **COPY**), then use the project fileset as well as the **PROJECT** parameter.

Versions 7

LIBRARIAN is a powerful configuration management tool that allows you to create baselines for your applications at specific strategic points in time. This chapter describes how to manage versions of applications within LIBRARIAN. Topics discussed in this chapter include:

- Version Management
- Working with Versions
- Using Forward Versioning
- Tags
- Reviewing Version and Tag Information

Version Management

A *version* is a collection of files in an application at a selected point in time, corresponding to a particular release or configuration of the software. Establishing versions for your applications lets you:

- manage the files in a version as a single entity,
- group different kinds of related files (for example, source code, executables, graphics, documentation, etc.),
- track changes to applications, and
- distribute versions easily and quickly.

Versions are often used to identify a software release. In a development or maintenance environment, several versions of an application often coexist. LIBRARIAN's version management capabilities allow you to define a version for an application at any point in time with a name you provide. You can then easily distribute or branch-off development of any existing version.

When you create a version, the current revision of each file in the application's master library is stamped with the version identifier. Members of a version can be retrieved individually or as a whole, even when newer revisions or versions are introduced. You typically retrieve files from previous versions for modification, distribution, or restoration.

Note *Versions* apply to an entire application; *revisions* refer to changes made to individual files within an application.

The initial revision of each file in an application comprising a version is the *base revision* or *root revision* of that file. The version count (**VCOUNT**) for the base revision is 0, each time you check in a revision this counter is incremented by 1. Base revisions are protected from being flushed until you specify that the version is obsolete. Refer to Chapter 4, "Revisions", in the *LIBRARIAN/iX User's Guide* for more information about managing revisions to files.

LIBRARIAN version management allows you to describe and reference all files comprising an application at a specific point in time. With versions, revisions associated with the version make up a baseline and are protected as a collection of files.

Figure 7-1 shows two versions of a sample application that includes five files. The first version is called REL-1.

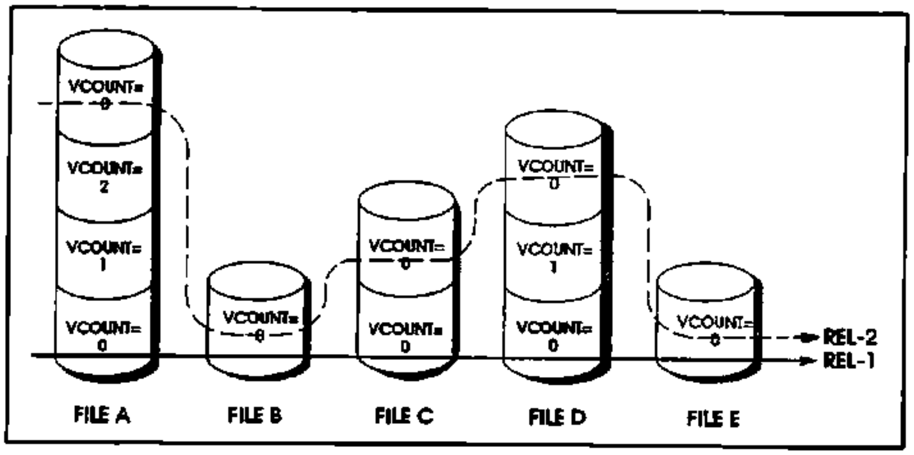


Figure 7-1. Sample Versions

During the revision process, file A is modified three times, and three generations are retained. File C was modified once, and file D was modified twice. Files B and E were not modified.

When new version REL-2 is created, LIBRARIAN locates and marks the most current revision of each file in the application. These files comprise the new base version and the **VCOUNT** is set to 0. Because files B and E were not modified, they are members of both REL-1 and REL-2. Files that are members of current and previous versions cannot be flushed unless all versions to which the file belongs are obsolete. The version created is REL-1 and the current version is REL-2.

Base revision files for REL-1 remain protected. These files are kept until you specify the version as obsolete by using the **OBSOLETE** parameter of the **VERSION** command. Any revisions expired or associated with an obsolete version are flushed using the FLUSH utility.

Working with Versions

LIBRARIAN tracks the various versions of applications with user defined version IDs. When you create a version, LIBRARIAN prefixes all revision identifiers with the new version ID. For example, REL2:0 indicates the base revision of REL2. If no version has ever been defined for an application, the version ID is an asterisk (*).

Creating Versions

Create the first version when defining the file library for the application, and before the first file is checked out of the library. Record the version using the **VERSION** command. If your files are not ready for release, you could call the initial version a pre–release.

For example, stamp every master file in the PAYR application with the version ID PREREL-1 by typing:

```
>VERSION PAYR ;ID=PREREL-1 ;DESCRIPTION=prerelease 1
```

You can also create a version by selecting the version option from the Admin menu.

You should create versions of an application when releasing the entire application into production, or when creating a baseline for future development.

Referring to Versions in LIBRARIAN

A user with LIBRARIAN Manager, Application Manager, or Operator capability can use the **RESTORE** command to restore a previous revision of a single file or a version of an entire application from the library. In menu mode, you can specify a version in dialogs that restore and/or copy files by pressing the "Revision Criteria" function key. These options are also available in command mode. For example, the following command restores the revision of a file that was part of the REL–2 version baseline:

MPE
```
>RESTORE REL-2 OF DREP.PUB.FIN
```

UNIX
```
>RESTORE REL-2 OF /apps /finance/pub/drep
```

In addition, all of the LIBRARIAN commands that perform file operations can use version references. All references to previous versions retrieve the base revision unless the **VCOUNT** is included or revision ID is specified. The following command copies the version REL-5 files, using baseline to a test:

MPE
```
>COPY REL-5 OF %PAYFILES TO =.TEST5
```

UNIX
```
>COPY REL-5 OF %PAYFILES TO ./test5/=
```

If you are retaining intermediate revisions of files, you can retrieve them by referencing the version and the **VCOUNT** or revision ID. For example, the following command copies the second revision of PAYTAB.PUB.FIN in REL-2:

MPE
```
>COPY REL-2 OF PAYTAB.PUB.FIN;VCOUNT=2
```

UNIX
```
>COPY REL-2 OF /fin/pub/paytab;VCOUNT=2
```

You can check out or distribute a version using defined steps. For example:

```
>PAYR-DIST REL-2 OF %PAYROLL
```

Changing Version Status

You might want to declare some versions obsolete and flush retained base version files.

Make a version obsolete by using the **OBSOLETE** parameter of the **VERSION** command or selecting the Version option from the **Admin** menu. For example, make the first pre–release version of the PAYR application obsolete by typing:

```
>VERSION PAYR ;ID=PREREL-1 ;OBSOLETE
```

Return an obsolete version to its previous status using the **UNOBSOLETE** parameter of the **VERSION** command. For example, unobsolete the first pre–release version of PAYR1 by typing:

```
>VERSION PAYR ;ID=PREREL-1 ;UNOBSOLETE
```

Note You cannot make a version obsolete if an active older version exists; you cannot reinstate a version if an obsolete newer version exists.

Deleting Versions

It is not necessary to delete flushed versions. It is recommended that you keep old version records as an audit trail of the application's version history.

To delete a version, use the **DELETE** parameter of the **VERSION** command. For example, delete the pre–release version of the PAYR application by typing:

```
>VERSION PAYR ;ID=PREREL-1 ;DELETE
```

Note A version must be marked as obsolete and the files flushed using the **FLUSH** utility prior to deleting the version using the **DELETE** parameter of the **VERSION** command.

Using Forward Versioning

LIBRARIAN provides an alternate method for managing successive releases of an application called *Forward Versioning*. This method requires storing each major version of an application in a separate location. Files are checked out from the *old* location, modified, compiled, and checked into the *new* location.

Thus, the new location is gradually built up until the new version is complete and ready for distribution to one or more production locations. Each version can include all of the files in the application, or you can choose to include only the programs that have changed.

How Forward Versioning Works

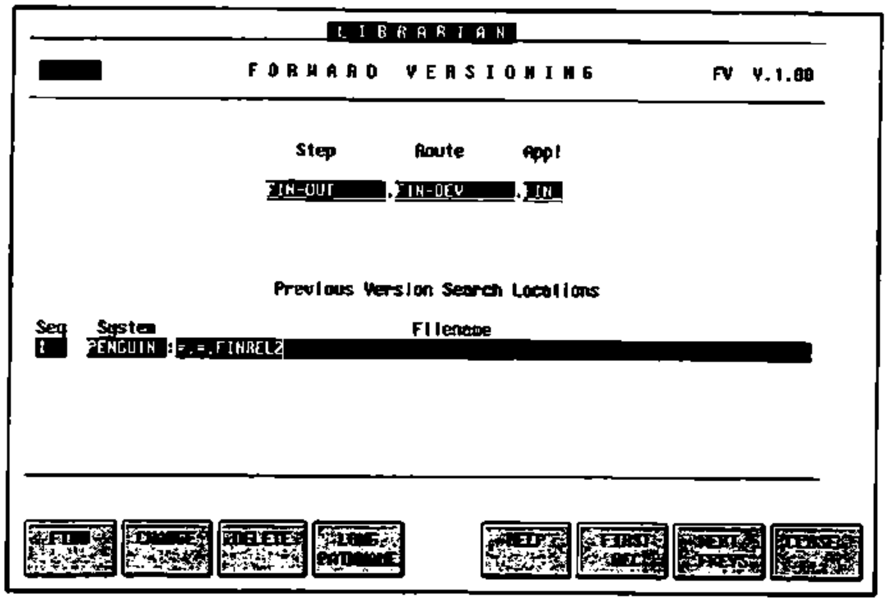
When you use a checkout step that has forward versioning rules associated with it, LIBRARIAN first attempts to check out file(s) from the *new* version location. This location is specified as the source location on the Steps (ST) screen.

If the file(s) do not exist in that location, then LIBRARIAN searches for the file(s) in the alternate (*old*) version location(s) as defined on the Forward Versioning (FV) screen. Pending master records are created for files that are checked out from an *old* location, so that they will automatically be moved to the *new* location on check–in.

Setting Up Forward Versioning

You define forward versioning rules for the checkout (master–to–secondary) step using the ST and FV screens. No modification to the checkin step is required. To set up forward versioning, do the following:

1. Define both the *old* and the *new* master version locations within the same application. Further, make sure that these locations are in the scope of the step fileset. Then, use SHORTCUT, or use the Auto Filesets (AF) screen followed by running **AUTOUPDATE**.
2. Set up the checkout step using SHORTCUT or the Steps (ST) screen to copy files from the *new* version location to the development area, even though no files exist yet in the *new* location.
3. Define forward versioning rules for the checkout step on the Forward Versioning (FV) screen. Enter the *old* library location(s) as alternate search location(s). See Figure 7–2 for an example.



```
                              LIBRARIAN
              F O R W A R D   V E R S I O N I N G          FV V.1.00

                        Step        Route       Appl
                      FIN-OUT    . FIN-DEV    . FIN

                  Previous Version Search Locations
Seq  System                       Filename
1    PENGUIN  : =.=.FINREL2
```

Figure 7-2. Forward Versioning (FV) Screen

Multiple Search Locations

It is possible to define multiple alternate search in multiple locations for a single checkout step. The sequence of alternate location checking is determined by the sequence number specified on the FV screen. Using this approach, it is possible to set up a "base release" location, then separate locations for each subsequent "minor" or "partial" release. The FV checkout searches each location in turn until it finds the requested file(s), always retrieving the latest version of the file(s). This is illustrated in Figure 7-3.

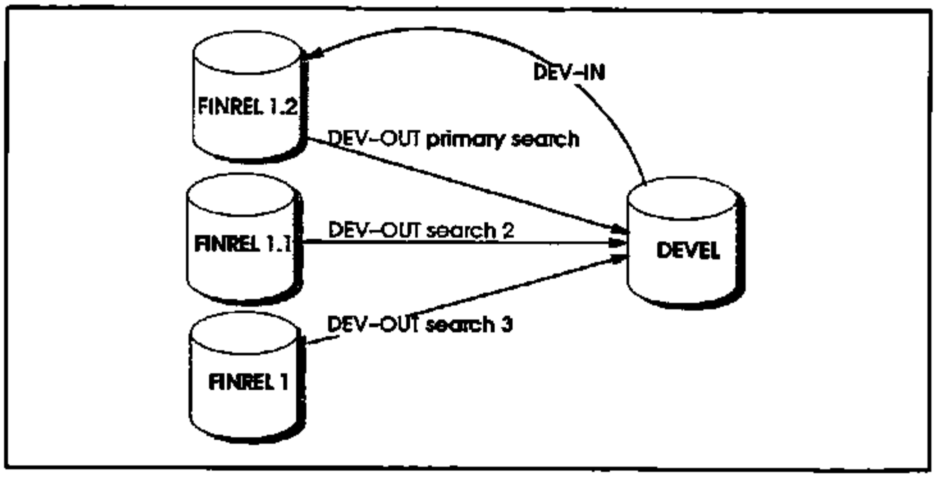


Figure 7-3. Alternate Search Locations for New Release

A new FV location must be added and the ST source location changed every time a *new* release account is created. To establish a new "base release," simply check out the entire application and check it in to a new directory, purge all the old directories, and start the process over.

Using Forward Versioning for File Distribution

Distribution steps (master–to–secondary, read mode) can also make use of forward versioning. In the multi–account FV model described in the preceding section, an FV step can be used to distribute the entire application's executable files, and will always select the most current version of each file.

Concurrent Maintenance and Development

Forward Versioning can be used to support concurrent, or overlapping maintenance and development. In the scenario described above, the FV checkout is used for development, creating a new library location with the changed modules only. To allow maintenance of the *old* version while development is taking place, set up a second checkout–checkin process, possibly in a different route, to check–out from and check–in to the *old* location.

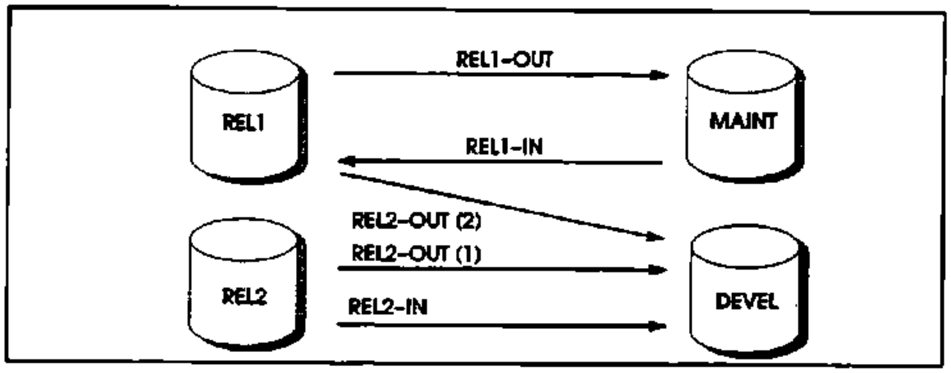


Figure 7–4. Concurrent Maintenance and Development

With this process, it is possible for one programmer to maintain a given program through the maintenance route while another programmer enhances the same program for the next release through the forward versioning route. Both copies of the file are write mode secondaries. Unlike the "emergency fix" (**PUSHREAD**) approach to this situation, there is no notification or exception logic when either secondary is checked in. Such notification and lock can be achieved with a macro, if desired.

Using Forward Versioning with Vendor Software

Forward Versioning is useful for managing custom changes to vendor–supplied software by allowing you to keep the vendor's original source physically separate from your customized changes.

To accomplish this, set up forward versioning as described above and restore the unmodified vendor source into the *old* location. Use the FV checkout to make local changes, checking in to the *new* location. Under this scheme, programmers do not need to know whether a given program has been customized or not. If it has, it will be checked out from the *new* location; if not, they will get the original vendor source from the *old* location.

To install a new vendor release, restore files into the *old* location or restore into a separate location and checkin to the *old* location. Check out each of the customized files from the *new* location, integrate vendor changes, and check in.

Tags

In addition to identifying entire applications with version identifiers, you can assign a tag to any set of files within an application for future reference. One example of using tags is to identify a subset of an application's files that make up a patch for distribution.

To assign a tag to a group of master files, use the **SET TAG** command or select **Set...Tag** from the **File** menu. For example:

```
>SET ABC@.SOURCE.PROD; TAG=PATCH201
```

```
>SET /prod/source/abc*; TAG=PATCH201
```

Regardless of how many times these files are revised in the future you can always refer to the correct revisions that made up this patch by typing (or using the revision criteria option in menu mode):

```
>A-OUT ABC100,ABC200 ;TAG=PATCH201
```

LIBRARIAN checks out the tagged revision of each file, branching if necessary. See Chapter 4, "Revisions", in the *LIBRARIAN/iX User's Guide*.

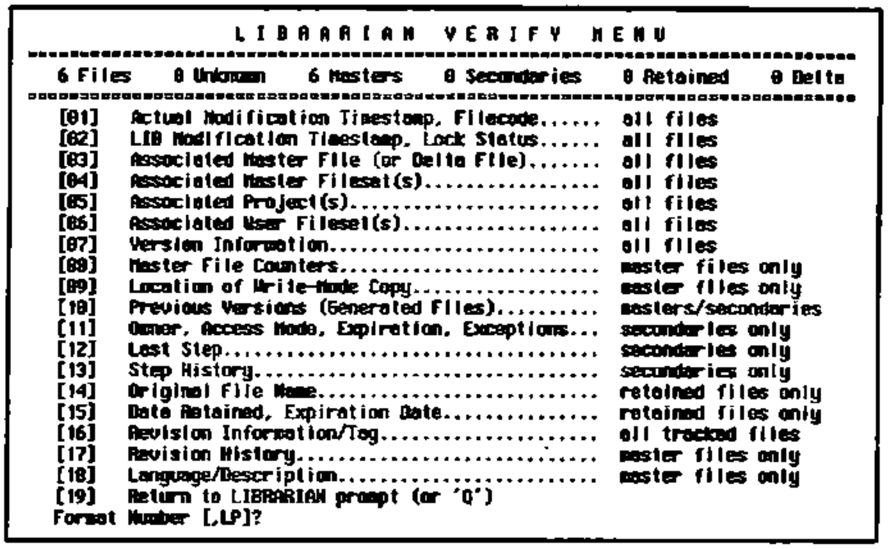
You can display the tags for files using the **VERIFY** command, format 16, "Revision Information".

Reviewing Version and Tag Information

You can review version and tag information by using the **VERIFY** command. For example, to view information for the files in the ABC application, type:

```
>VERIFY %ABC
```

VERIFY produces the menu shown in Figure 7–5.

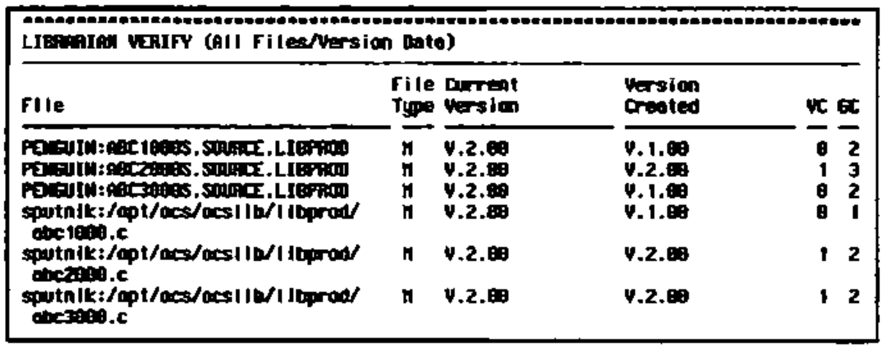


```
                 L I B R A R I A N   V E R I F Y   M E N U
===============================================================================
  6 Files    0 Unknown    6 Masters    0 Secondaries    0 Retained    0 Delta
===============================================================================
   [01]   Actual Modification Timestamp, Filecode......  all files
   [02]   LIB Modification Timestamp, Lock Status......  all files
   [03]   Associated Master File (or Delta File).......  all files
   [04]   Associated Master Fileset(s).................  all files
   [05]   Associated Project(s)........................  all files
   [06]   Associated User Fileset(s)...................  all files
   [07]   Version Information..........................  all files
   [08]   Master File Counters.........................  master files only
   [09]   Location of Write-Mode Copy..................  master files only
   [10]   Previous Versions (Generated Files)..........  masters/secondaries
   [11]   Owner, Access Mode, Expiration, Exceptions...  secondaries only
   [12]   Last Step....................................  secondaries only
   [13]   Step History.................................  secondaries only
   [14]   Original File Name...........................  retained files only
   [15]   Date Retained, Expiration Date...............  retained files only
   [16]   Revision Information/Tag.....................  all tracked files
   [17]   Revision History.............................  master files only
   [18]   Language/Description.........................  master files only
   [19]   Return to LIBRARIAN prompt (or 'Q')
  Format Number [,LP]?
```

Figure 7–5. VERIFY Menu

Format 7 displays version information. Format 16 displays tag information.

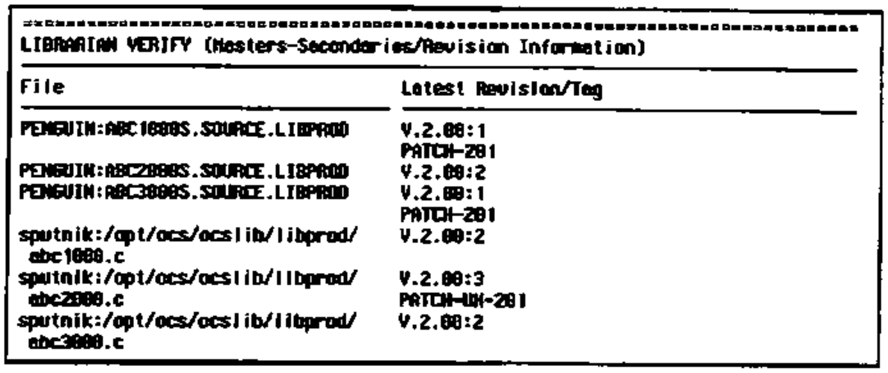
Figure 7–6 shows format 7, version data, for the PAYR application.



```
===============================================================================
LIBRARIAN VERIFY (All Files/Version Date)
-------------------------------------------------------------------------------
                                   File Current            Version
File                               Type Version            Created         VC GC
---------------------------------- ---- ----------------   --------------- -- --
PENGUIN:ABC1000S.SOURCE.LIBPROD     M   V.2.00             V.1.00           0  2
PENGUIN:ABC2000S.SOURCE.LIBPROD     M   V.2.00             V.2.00           1  3
PENGUIN:ABC3000S.SOURCE.LIBPROD     M   V.2.00             V.1.00           0  2
sputnik:/opt/ocs/ocslib/libprod/    M   V.2.00             V.1.00           0  1
 abc1000.c
sputnik:/opt/ocs/ocslib/libprod/    M   V.2.00             V.2.00           1  2
 abc2000.c
sputnik:/opt/ocs/ocslib/libprod/    M   V.2.00             V.2.00           1  2
 abc3000.c
```

Figure 7–6. VERIFY Display with Version Data (Format 7)

Figure 7–7 shows the tags for the ABC application using format 16.



```
==========================================================================
LIBRARIAN VERIFY (Masters-Secondaries/Revision Information)
--------------------------------------------------------------------------
File                                 Latest Revision/Tag
---------------------------------    ---------------------------------
PENGUIN:ABC1000S.SOURCE.LIBPROD      V.2.00:1
                                     PATCH-201
PENGUIN:ABC2000S.SOURCE.LIBPROD      V.2.00:2
PENGUIN:ABC3000S.SOURCE.LIBPROD      V.2.00:1
                                     PATCH-201
sputnik:/opt/ocs/ocslib/libprod/     V.2.00:2
 abc1000.c
sputnik:/opt/ocs/ocslib/libprod/     V.2.00:3
 abc2000.c                           PATCH-UX-201
sputnik:/opt/ocs/ocslib/libprod/     V.2.00:2
 abc3000.c
```

Figure 7–7. VERIFY Display Showing Tags (Format 16)

Version Reports

Version information is available in standard LIBRARIAN reports, as shown in Table 7–1.

Table 7–1. Version Information in Standard Reports

Report Code	Report Title	Description
RFN10	Pre-Flush Notification	Files that will be flushed by user and by filename.
RFN20	Pre-Flush Notification	Files that will be flushed by filename.
FLUSH	Flush Utility Detail	Files that were flushed.
RAV10	Application Version	Versions for an application.
RFD10	Fileset Status	Version membership for specific files.
RFD20	Master File Status	All master files and associated files.
RGF10	Generated Files	Cross reference of retained generations and original filenames.
RRH10	Revision History	Shows history of file revisions with timestamps.
RVD10	File Version	Shows detailed information on all files in a version.

Reports 8

LIBRARIAN's reports and online inquiries allow you to review the rules you have defined, file status and history information, and the audit trail records.

This chapter describes how to generate reports and online inquiries. Topics discussed in this chapter include:

- Generating Reports
- File Inquiry Screen
- Project Inquiry Screen
- File Information Using **VERIFY**
- Version Inquiries
- Step/project Inquiries Using **HELP**
- The SHOWLOG Report Writer

Generating Reports

You can review your rules at any time by generating standard reports. These reports reflect the rules you defined for the library, file movements, and user authorizations, in addition to the current status of files, filesets, projects, versions, and listings of files ready to be flushed.

Menu Mode

You can generate any of the standard reports by selecting **Files...**, **Versions...**, **Rules...**, or **Log...** from the **Info** menu. Select the report you want from the appropriate menu. Some reports will require answers to a series of prompts.

Command Mode

You can also generate any of the standard reports directly from the LIBRARIAN prompt. Type the report name and press ENTER. For example:

```
>RFX10
```

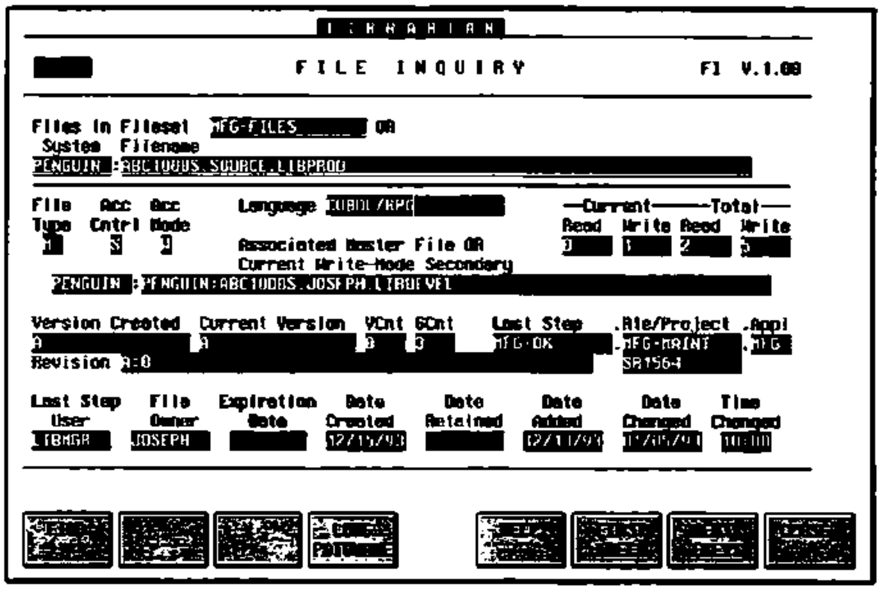
In addition, you can generate standard reports from report jobstreams, and redefine the output device and priority.

For more information on reports, refer to Chapter 6, "Reports", in the *LIBRARIAN/iX Reference Guide.*

File Inquiry Screen

The File Inquiry (FI) screen provides complete status information for a file or fileset, including the access mode, related master or secondary files, historical totals for read mode and write mode access, version membership information, most recent transaction information, and dates of file creation, retention, and expiration. To access this screen, select **Files...** from the **Info** menu and select the **FI** option. From the command line, type **FI** followed by ENTER.

Figure 8–1 shows a file inquiry for a master file on one system with a write mode copy in development. This information is also available with the **VERIFY** command.

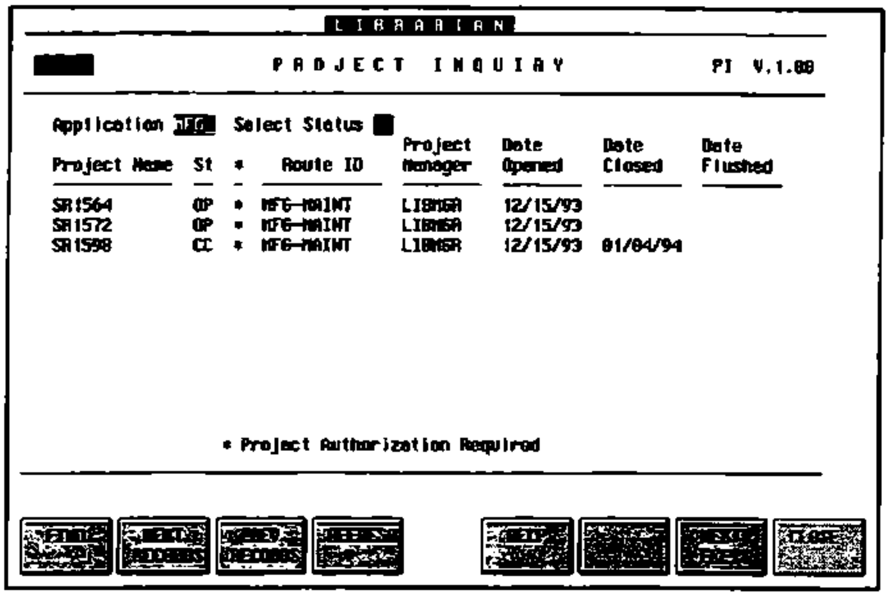


```
                              L I B R A R I A N
                           F I L E   I N Q U I R Y                     FI  V.1.00

Files in Fileset  MFG-FILES_______ OR
 System   Filename
PENGUIN  :ABC1000S.SOURCE.LIBPROD

File   Acc   Acc      Language COBOL/RPG              —Current———Total—
Type   Cntrl Mode                                      Read  Write Read  Write
 M      S     D       Associated Master File OR       0     1     2     5
                      Current Write-Mode Secondary
  PENGUIN  :PENGUIN:ABC1000S.JOSEPH.LIBDEVEL

Version Created  Current Version  VCnt GCnt   Last Step    .Ale/Project .Appl
A                A                0    0      MFG-OK       .MFG-MAINT   .MFG
Revision A:0                                               SR1564

Last Step  File    Expiration  Date      Date      Date      Date      Time
  User     Owner     Date     Created   Retained   Added    Changed   Changed
LIBMGR    JOSEPH              12/15/93            12/11/93  11/05/93  10:00
```

Figure 8–1.   File Inquiry (FI) Screen

Project Inquiry Screen

Project inquiries provide information on the status of projects for an application. Figure 8–2 shows a project inquiry for all projects in the MFG application.



```
                          L I B R A R I A N
                 P R O J E C T   I N Q U I R Y              PI  V.1.00

  Application MFG  Select Status
                                     Project   Date      Date      Date
  Project Name  St  *    Route ID    Manager   Opened    Closed    Flushed
  ------------  --  -  ----------    -------   --------  --------  -------
  SR1564        OP  *  MFG-MAINT     LIBMGR    12/15/93
  SR1572        OP  *  MFG-MAINT     LIBMGR    12/15/93
  SR1598        CC  *  MFG-MAINT     LIBMGR    12/15/93  01/04/94

                * Project Authorization Required
```

Figure 8–2. Project Inquiry (PI) Screen

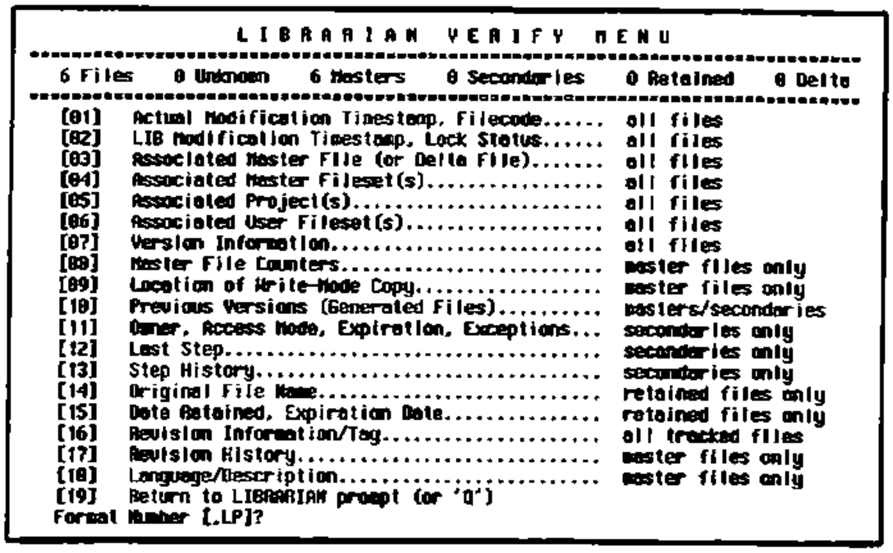
To access this screen choose **Projects...** from the **Screens** menu, from the **Admin** menu, or enter **PI** on the command line.

File Information Using VERIFY

The **VERIFY** command offers extensive information about files. You can review information such as who last checked out a file, when it was last checked out, where it was copied, and which step was performed. When issuing **VERIFY**, specify the files or filesets to review. For example:

```
>VERIFY %MFG-FILES, F@.PUB.FIN, J##P.JOB.FIN
```

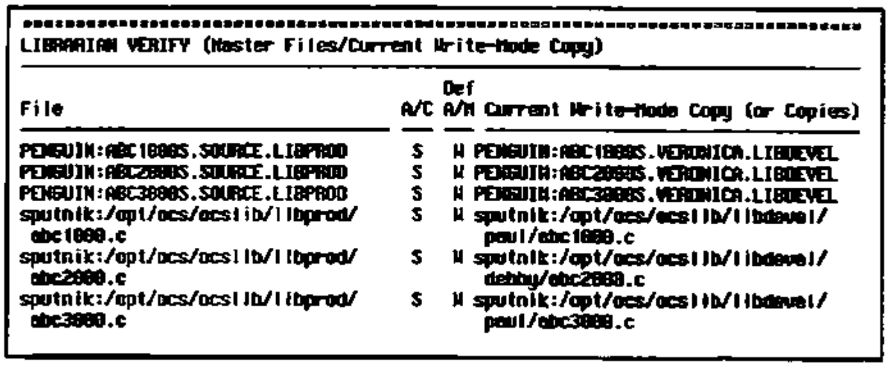
VERIFY displays a menu of formats. Each format includes different information. You can review the information online or use the **OFFLINE** (or **LP**) parameter to send the information to another device. Figure 8–3 shows the **VERIFY** menu.



```
                     L I B R A R I A N   V E R I F Y   M E N U
==================================================================================
   6 Files    0 Unknown    6 Masters    0 Secondaries    0 Retained    0 Delta
==================================================================================
  [01]   Actual Modification Timestamp, Filecode......  all files
  [02]   LIB Modification Timestamp, Lock Status......  all files
  [03]   Associated Master File (or Delta File).......  all files
  [04]   Associated Master Fileset(s).................  all files
  [05]   Associated Project(s)........................  all files
  [06]   Associated User Fileset(s)...................  all files
  [07]   Version Information..........................  all files
  [08]   Master File Counters.........................  master files only
  [09]   Location of Write-Mode Copy..................  master files only
  [10]   Previous Versions (Generated Files)..........  masters/secondaries
  [11]   Owner, Access Mode, Expiration, Exceptions...  secondaries only
  [12]   Last Step....................................  secondaries only
  [13]   Step History.................................  secondaries only
  [14]   Original File Name...........................  retained files only
  [15]   Date Retained, Expiration Date...............  retained files only
  [16]   Revision Information/Tag.....................  all tracked files
  [17]   Revision History.............................  master files only
  [18]   Language/Description.........................  master files only
  [19]   Return to LIBRARIAN prompt (or 'Q')
  Format Number [,LP]?
```

Figure 8–3. VERIFY Menu

When you enter a format number, the requested information appears. You can continue to choose different formats for the same files until you type Q to quit and return to the LIBRARIAN prompt. Figure 8–4 is a sample display of master files and the location of their associated write mode copies (format 9).

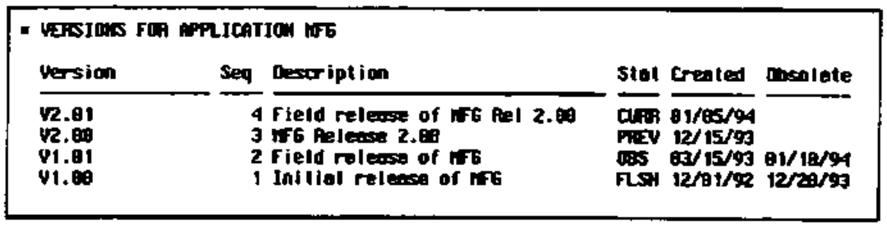


```
==================================================================================
LIBRARIAN VERIFY (Master Files/Current Write-Mode Copy)
__________________________________________________________________________________
                                          Def
File                                  A/C A/M Current Write-Mode Copy (or Copies)
_____________________________________ ___ ___ ____________________________________
PENGUIN:ABC1000S.SOURCE.LIBPROD        S   W PENGUIN:ABC1000S.VERONICA.LIBDEVEL
PENGUIN:ABC2000S.SOURCE.LIBPROD        S   W PENGUIN:ABC2000S.VERONICA.LIBDEVEL
PENGUIN:ABC3000S.SOURCE.LIBPROD        S   W PENGUIN:ABC3000S.VERONICA.LIBDEVEL
sputnik:/opt/ocs/ocslib/libprod/       S   W sputnik:/opt/ocs/ocslib/libdevel/
 abc1000.c                                     paul/abc1000.c
sputnik:/opt/ocs/ocslib/libprod/       S   W sputnik:/opt/ocs/ocslib/libdevel/
 abc2000.c                                     debby/abc2000.c
sputnik:/opt/ocs/ocslib/libprod/       S   W sputnik:/opt/ocs/ocslib/libdevel/
 abc3000.c                                     paul/abc3000.c
```

Figure 8–4. VERIFY Display (Format 9)

Version Inquiries

The **VERSION** command (available from the **Admin** menu) provides information on all of the versions for an application, including current status and baseline dates. The following shows a version inquiry for the AP application. The same information is available offline in the Versions Report (RAV10).



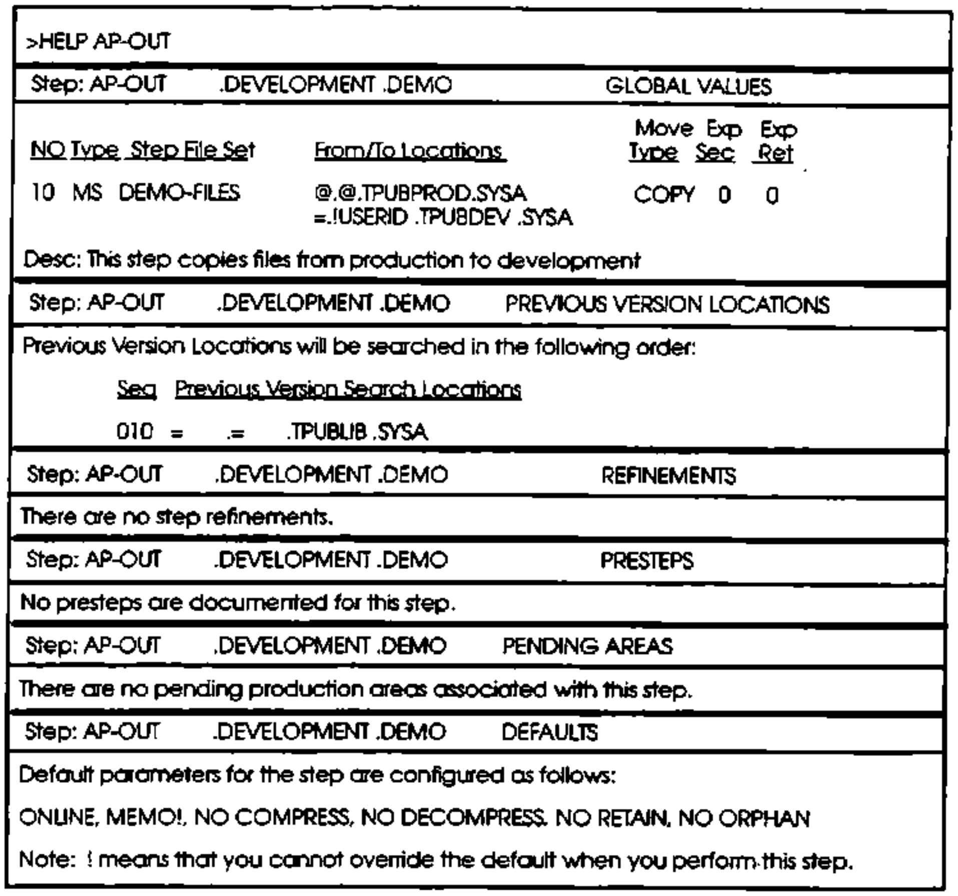
```
■ VERSIONS FOR APPLICATION MFG

  Version          Seq  Description                        Stat Created  Obsolete
  ---------------  ---  ---------------------------------  ---- -------- --------
  V2.01              4 Field release of MFG Rel 2.00       CURR 01/05/94
  V2.00              3 MFG Release 2.00                    PREV 12/15/93
  V1.01              2 Field release of MFG                OBS  03/15/93 01/10/94
  V1.00              1 Initial release of MFG              FLSH 12/01/92 12/20/93
```

Figure 8–5 Versions Display

Step/Project Inquiries

You can review step and project information online with the **HELP** command, using the stepname as its parameter, you can request information on the defined defaults, allowed overrides, and other step information. You can also obtain this information for steps by pressing **F1** on the **Steps** menu (accessed from the **File** menu) or for projects by opening the **Projects** menu available from the **Settings** window (accessed from the **User** menu).

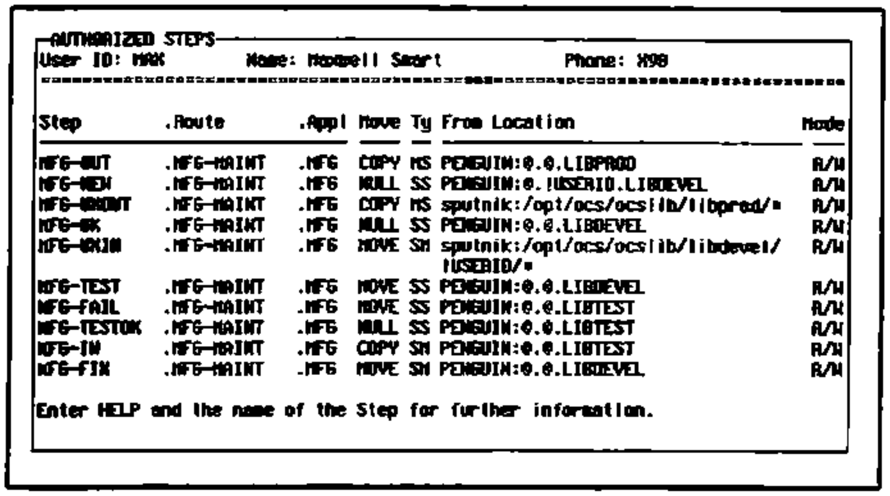
Figure 8–6 shows the step information windows for the AP-OUT step.



```
>HELP AP-OUT
Step: AP-OUT     .DEVELOPMENT .DEMO               GLOBAL VALUES

                                                    Move  Exp  Exp
NO Type  Step File Set     From/To Locations        Type  Sec  Ret

10  MS   DEMO-FILES        @.@.TPUBPROD.SYSA        COPY   0    0
                           =.!USERID .TPUBDEV .SYSA

Desc: This step copies files from production to development
Step: AP-OUT     .DEVELOPMENT .DEMO     PREVIOUS VERSION LOCATIONS
Previous Version Locations will be searched in the following order:

         Seq  Previous Version Search Locations

         010  =     .=     .TPUBLIB .SYSA
Step: AP-OUT     .DEVELOPMENT .DEMO               REFINEMENTS
There are no step refinements.
Step: AP-OUT     .DEVELOPMENT .DEMO               PRESTEPS
No presteps are documented for this step.
Step: AP-OUT     .DEVELOPMENT .DEMO     PENDING AREAS
There are no pending production areas associated with this step.
Step: AP-OUT     .DEVELOPMENT .DEMO     DEFAULTS
Default parameters for the step are configured as follows:
ONLINE, MEMO!, NO COMPRESS, NO DECOMPRESS, NO RETAIN, NO ORPHAN
Note: ! means that you cannot override the default when you perform this step.
```

Figure 8–6. Help Information for AP-OUT Step

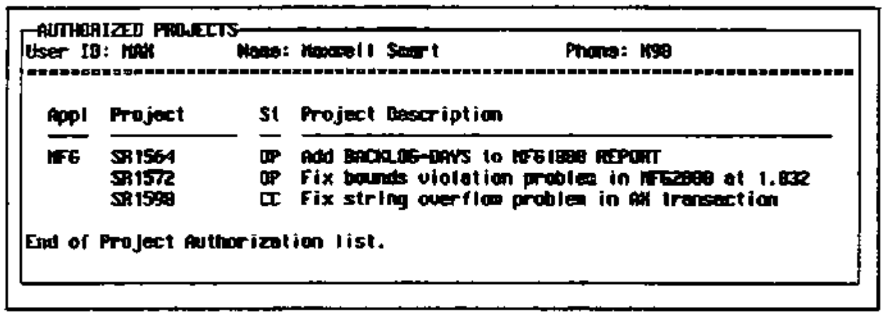
With the **HELP STEPS** command, you can review a list of the steps you are authorized to perform and information about the step, as shown in Figure 8–7.



```
AUTHORIZED STEPS
User ID: MAX          Name: Maxwell Smart              Phone: X98
==========================================================================

Step        .Route        .Appl Move Ty From Location                        Mode
_____________________________  ____ __ ___________________________________   ____
MFG-OUT     .MFG-MAINT    .MFG  COPY MS PENGUIN:0.0.LIBPROD                  R/W
MFG-NEW     .MFG-MAINT    .MFG  NULL SS PENGUIN:0.!USERID.LIBDEVEL           R/W
MFG-UNOUT   .MFG-MAINT    .MFG  COPY MS sputnik:/opt/ocs/ocslib/libprod/*    R/W
MFG-OK      .MFG-MAINT    .MFG  NULL SS PENGUIN:0.0.LIBDEVEL                 R/W
MFG-UNIN    .MFG-MAINT    .MFG  MOVE SM sputnik:/opt/ocs/ocslib/libdevel/    R/W
                                        !USERID/*
MFG-TEST    .MFG-MAINT    .MFG  MOVE SS PENGUIN:0.0.LIBDEVEL                 R/W
MFG-FAIL    .MFG-MAINT    .MFG  MOVE SS PENGUIN:0.0.LIBTEST                  R/W
MFG-TESTOK  .MFG-MAINT    .MFG  NULL SS PENGUIN:0.0.LIBTEST                  R/W
MFG-IN      .MFG-MAINT    .MFG  COPY SM PENGUIN:0.0.LIBTEST                  R/W
MFG-FIX     .MFG-MAINT    .MFG  MOVE SM PENGUIN:0.0.LIBDEVEL                 R/W

Enter HELP and the name of the Step for further information.
```

Figure 8–7. Help Steps

In addition, you can use the **HELP PROJECTS** command to review an online list of which projects you are authorized to work on, as shown in Figure 8–8.



```
AUTHORIZED PROJECTS
User ID: MAX          Name: Maxwell Smart              Phone: X98
==========================================================================

  Appl  Project      St  Project Description
  ____  ___________  __  _________________________________________________
  MFG   SR1564       OP  Add BACKLOG-DAYS to MFG1000 REPORT
        SR1572       OP  Fix bounds violation problem in MFG2000 at 1.032
        SR1598       CC  Fix string overflow problem in AX transaction

End of Project Authorization list.
```

Figure 8–8. Help Projects

Using the SHOWLOG Report Writer

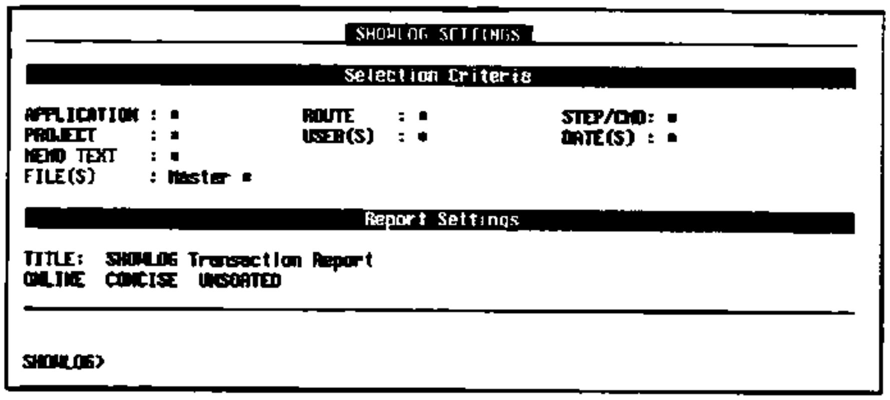
SHOWLOG is a flexible report writer that allows you to select and report transaction data from the LIBLOG audit trail database. By using SHOWLOG, you can review transaction history information selectively in a variety of formats.

In addition to reporting capabilities, SHOWLOG offers several other functions:

- **EDITMEMO** allows you to edit memos entered when performing steps.
- **LIST** allows you to create a listfile of files in selected transactions.
- **FLUSH** flushes selected log records.

Access SHOWLOG from LIBRARIAN with the SHOWLOG command or select **Log...** from the **Info** menu and then select SHOWLOG.

SHOWLOG initially displays default selection criteria and default report settings, followed by the SHOWLOG prompt, as shown in Figure 8–9. The default is to include all transactions in the database. The default report settings are the concise format, unsorted order, and an online display.



```
                         SHOWLOG SETTINGS
                        Selection Criteria
APPLICATION : *          ROUTE    : *             STEP/CMD: *
PROJECT     : *          USER(S)  : *             DATE(S) : *
MEMO TEXT   : *
FILE(S)     : Master *
                         Report Settings
TITLE:  SHOWLOG Transaction Report
ONLINE  CONCISE  UNSORTED

SHOWLOG>
```

Figure 8–9. Initial SHOWLOG Display

Use SHOWLOG commands listed in Table 8–1 to specify the transactions you want to review and the way you want them presented. For more information, refer to Chapter 4, "SHOWLOG Commands" in the *LIBRARIAN/iX Reference Guide*.

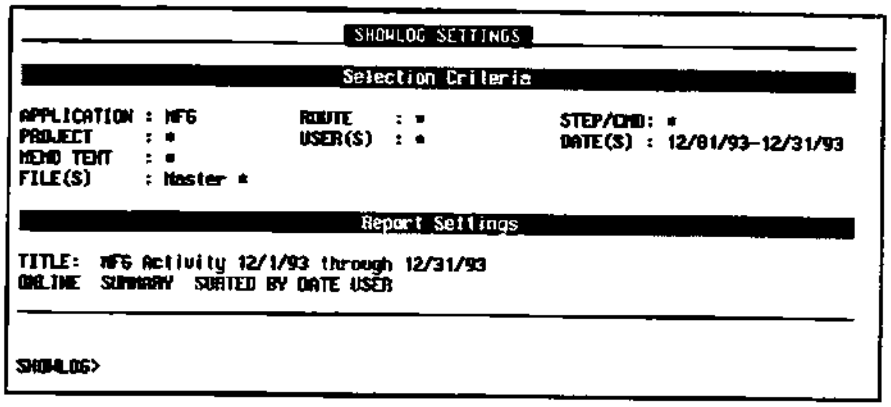
Table 8-1. SHOWLOG Commands Summary

Command	Function
Selection Criteria	
SHOWLOG>SE[LECT]	Extracts only those transaction records which match the specified selection criteria.
Report Format	
SHOWLOG>FO[RMAT]	Changes report format.
SHOWLOG>LI[ST]	Creates a listfile containing the names of files involved in selected transactions.
Output Definition	
SHOWLOG>OU[TPUT]	Sets the report output disposition to offline or online.
Sort Sequence	
SHOWLOG>SO[RT]	Sets up the report sort sequence.
Report Settings	
SHOWLOG>GE[T]	Processes commands from a file.
SHOWLOG>SA[VE]	Saves current report settings and selection criteria in a file.
Generate Reports	
SHOWLOG>GO	Generates report using current report selection criteria and settings.
Subsets	
SHOWLOG>SUB[SET]	Selects a subset of currently extracted transactions for reporting.
SHOWLOG>UN[DO]	Resets the current subset.
Other Commands	
SHOWLOG>EX[IT]	Terminates an active **SHOWLOG** session, and returns you to the LIBRARIAN prompt.
SHOWLOG>FL[USH]	Deletes all log records associates with extracted transactions.
SHOWLOG>HE[LP]	Accesses the online help for information about using **SHOWLOG**.
SHOWLOG>RED[O]	Edits the previous command entry.
SHOWLOG>RES[ET]	Resets selection criteria and/or report settings to default values.
SHOWLOG>SH[OW]	Displays selection criteria and report settings.
SHOWLOG>TI[TLE]	Sets a title to appear on all pages of a report.

The following sequence of commands selects transactions for all steps performed by any user in the MFG application between December 1st and 31st 1993, and selects the Summary format, sorted by date and user, with a new title.

```
SHOWLOG>APPLICATION MFG
SHOWLOG>12/1/93 - 12/31/93
SHOWLOG>SUMMARY
SHOWLOG>SORT DATE, USER
SHOWLOG>TITLE MFG Activity 12/1/93 through 12/31/93
```

Figure 8–10 shows the SHOWLOG display after you issue the above commands.

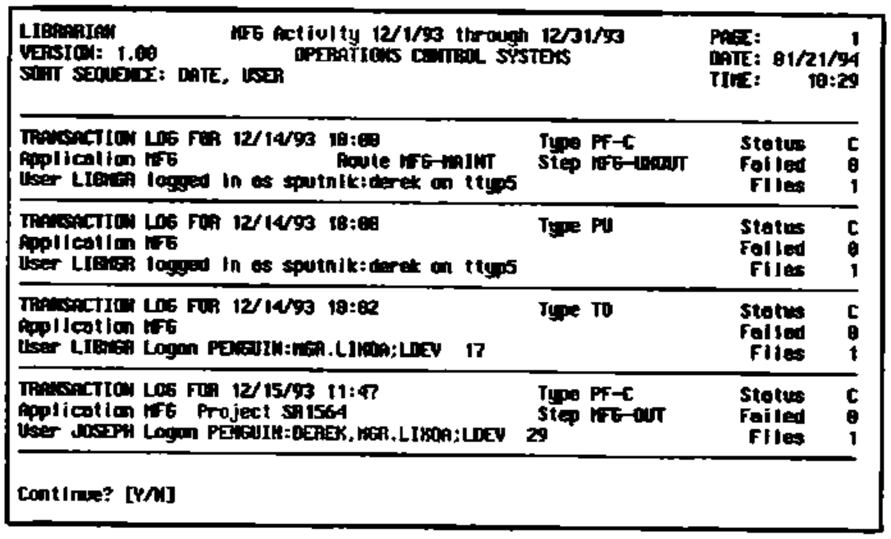


```
                              SHOWLOG SETTINGS
                              Selection Criteria
APPLICATION : MFG          ROUTE    : *          STEP/CMD: *
PROJECT     : *            USER(S)  : *          DATE(S) : 12/01/93-12/31/93
MEMO TEXT   : *
FILE(S)     : Master *
                              Report Settings
TITLE:  MFG Activity 12/1/93 through 12/31/93
ONLINE  SUMMARY  SORTED BY DATE USER

SHOWLOG>
```

Figure 8-10. SHOWLOG Display

After setting the selection, format, and output criteria, generate the report with **SHOWLOG>GO**. The report displays on the screen or is sent to the defined offline device.

Figure 8–11 contains a sample SHOWLOG summary report.



```
LIBRARIAN            MFG Activity 12/1/93 through 12/31/93        PAGE:        1
VERSION: 1.00              OPERATIONS CONTROL SYSTEMS             DATE: 01/21/94
SORT SEQUENCE: DATE, USER                                         TIME:    10:29

TRANSACTION LOG FOR 12/14/93 10:00                 Type PF-C          Status    C
Application MFG                 Route MFG-MAINT    Step MFG-UNOUT     Failed    0
User LIBMGR logged in as sputnik:derek on ttyp5                        Files    1

TRANSACTION LOG FOR 12/14/93 10:00                 Type PU            Status    C
Application MFG                                                       Failed    0
User LIBMGR logged in as sputnik:derek on ttyp5                        Files    1

TRANSACTION LOG FOR 12/14/93 10:02                 Type TO            Status    C
Application MFG                                                       Failed    0
User LIBMGR Logon PENGUIN:MGR.LINDA;LDEV  17                           Files    1

TRANSACTION LOG FOR 12/15/93 11:47                 Type PF-C          Status    C
Application MFG  Project SR1564                    Step MFG-OUT       Failed    0
User JOSEPH Logon PENGUIN:DEREK,MGR.LINDA;LDEV  29                     Files    1

Continue? [Y/N]
```

Figure 8-11. SHOWLOG Summary Report

Housekeeping 9

This chapter covers periodic housekeeping activities that the LIBRARIAN Administrator typically performs. Other users might perform some of these activities depending on how you implement LIBRARIAN. Topics in this section include:

- Flushing Expired Files
- Flushing Expired Transactions from the Audit Trail (LIBLOG)
- LIBRARIAN Databases – Capacity Management
- Changing Database Passwords
- Changing Network Configuration and Remote System Logon Information

Flushing Expired Files

LIBRARIAN allows you to set up rules to automatically purge, or "flush", certain types of files based on age and other criteria:

- Read mode secondaries – can be assigned a retention period by the step that creates them, or an expiration date through the **SET EXPDATE** command. Once expired, these files are eligible to be purged by the FLUSH utility.
- Retained masters and retained secondaries – (also called "g–files", either kept as MPE files or as deltas), are purged by the FLUSH utility only if they meet all of the following criteria:
    - The current date is greater than the expiration date assigned via the "safety retained days" on the step that created them, or set by a **SET EXPDATE** command.
    - They exceed the "minimum number of generations" configured on the SP screen.
    - They are not "base revisions" (revisions with a zero VCOUNT) of an active version.

Expired files, as described above, are purged by the FLUSH utility—either by issuing the **FLUSH** command from the command prompt, or by selecting **Flush** from the **Admin** pull–down menu. The FLUSH utility has no parameters; it simply purges files according to the rules defined and the expiration dates of the eligible files. Prior to running FLUSH, you may want to use the Pre–Flush Notification Reports (RFN10 and RFN20) to preview a list of files that will be purged by the utility. If files appear that you do not want to purge, use **SET EXPDATE** to set a new expiration date for these files (or set them to not expire).

A common strategy with FLUSH is to use both the "safety retained days" and the "minimum generations" restrictions to ensure that you have sufficient history for both frequently and infrequently changed files. In most installations, it is sufficient to run FLUSH once a month. If you use versions, you may want to set the other controls more aggressively, since base revisions will always be preserved.

Flushing Expired Transactions from the Audit Trail (LIBLOG)

The FLUSHLOG utility flushes, or deletes, audit trail transactions that meet the following criteria:

- The transaction is older than the Audit Trail "Aging Policy" set (in days) on the System Profile (SP) screen.
- If the transaction was associated with a project, that project has a status of "flush pending" (FP).

Specific log records can also be purged through SHOWLOG, by using the SHOWLOG commands to select records, then issuing the **FLUSH** command.

LIBRARIAN Databases – Capacity Management

LIBRARIAN uses two TURBOIMAGE databases, LIBDB (rules and file tracking information) and LIBLOG (audit trail). Datasets in these databases must be resized to accommodate growth, just like any TURBOIMAGE application.

The **CHECKDB** command is provided to assist with capacity management by reporting dataset capacities and flagging those that exceed a threshold percentage full. The command can be issued directly,

```
CHECKDB (threshold)
```

or as an argument to the MAIL command,

```
MAIL user;CHECKDB=threshold
```

In the **MAIL** command, mail will only be sent to the specified user if there are datasets that exceed the threshold percentage full.

Changing Database Passwords

For security reasons, you may wish to change LIBRARIAN database passwords periodically. Since these passwords are embedded in certain LIBRARIAN programs, the programs must be changed whenever the passwords are changed. To do this, run the program CONFIGP.COMP in the LIBRARIAN account. See Appendix C of this manual for details. CONFIGP changes only the programs; you must change the database passwords themselves using DBUTIL or some other utility.

Changing Network Configuration and Remote System Logon Information

LIBRARIAN account (MPE only) and user passwords are used to connect to remote and receiver systems, and are encrypted in the LIBRARIAN database. They are maintained on two screens: Network Configuration (NC) and Systems (SY). The NC screen maintains default values and SY specifies connection and logon/login information for individual systems if they are different from those recorded on the NC screen.

Network addresses, network type (NS, DS, or UNIX), and dial-up information are also maintained on these screens, so they are also used when adding clients or receivers or changing your network configuration. See the Reference Guide "Screens" chapter for detailed explanation.

MPE and UNIX client systems also require passwords to connect to the LIBRARIAN server. On MPE clients, this logon and password information is maintained via the CONFIGP.COMP program. On UNIX clients, this configuration information is maintained with the command *ocslib –config.*

Appendix A
Automatic Decompression

MPE only

LIBRARIAN provides file compression for security and as a means to save disk space. To use compressed files, it is necessary to first decompress them. This process provides a version of the file that is identical to the original uncompressed version, and the file is available for immediate access. Typically, the file decompression process must be initiated from LIBRARIAN using the **DECOMPRESS** command or the **OCSDCMP** program. The user initiates this process on an ad hoc basis.

In addition, LIBRARIAN provides an automatic method of decompressing files, as needed. In brief, programs that call the file system routine **FOPEN** can be trapped by LIBRARIAN to first determine whether a file needs to be decompressed. If a file is stored in compressed format, it is decompressed prior to the actual call to **FOPEN**. This feature is virtually transparent to the application and user.

If the application only needs to read the compressed file, a separate routine is provided that decompresses the file temporarily.

This appendix describes how to use automatic decompression with LIBRARIAN.

Enabling Automatic Decompression

Use the following steps to enable automatic decompression for each application program or group of application programs.

Add PH Capability to Application Programs

All programs that use automatic decompression must have PH capability. A utility called **PROGCAPS.COMP.OCSLIB** can add this capability. When you run this utility, you are prompted for a program name. Program capabilities are displayed. Type **PH** (in uppercase) to add PH capability to the program (lowercase indicates that the capability is absent).

Because of the Privileged Mode and stack requirements, file decompression is handled by a separate process. A process for the program **DCMPRSS.COMP.OCSLIB** is created when decompression is necessary; thus the requirement of PH capability.

Set Up Group/Public SL or XL

For compatibility mode programs, set up a Group (**;LIB=G**) or Public (**;LIB=P**) Segmented Library (SL) for each application group or account that includes the special **FOPEN** trap. If no SL is currently being used, copy SLFOPEN.COMP.OCSLIB to the program group or PUB group where the application programs reside. Be sure to rename SLFOPEN to SL. (For temporary decompression, use SLFOPENT.COMP.OCSLIB). Remember to use the **;LIB=** parameter of the **RUN** command.

If your application currently uses a Group or Public SL, add LIBRARIAN's **FOPEN** trap to it using **SEGMENTER**, as shown below:

```
:SEGMENTER
-SL SL
-USL FOPENU.COMP.OCSLIB
-ADDSL OCSFOPEN
-E
```

If your application currently uses an XL, add LIBRARIAN's **FOPEN** trap to it using **LINKEDIT**, as shown below:

```
:LINKEDIT
LinkEd> XL XL
LinkEd> ADDXL FOPENO.COMP.OCSLIB
LinkEd> E
```

(Use the USL file FOPENTU.COMP.OCSLIB for temporary decompression).

Alternatively, copy the **OCSFOPEN** segment from **FOPEN** to your program's USL file, or RL and then **PREP** your executable program referencing the USL or RL.

Note

Because LIBRARIAN's **FOPEN** trap cannot coexist in the same SL with the system **FOPEN** routine, programs residing in the PUB group of the SYS account (e.g., EDITOR) must be moved to a new group (e.g., PUB2). Make sure that your alternate group has the same capabilities as PUB.SYS.

For native mode programs, use the XL file XLFOPEN.COMP.OCSLIB (or XLFOPENT.COMP.OCSLIB to decompress files temporarily when accessed). Alternatively, use **LINKEDIT** to add FOPENO.COMP.OCSLIB (or FOPENTO.COMP.OCSLIB) to an XL or RL; or link it with other native mode object files to create an executable program file.

Allocate DCMPRSS.COMP.OCSLIB

For faster loading, OCS recommends that you allocate the automatic decompression program, as shown below:

```
:ALLOCATE DCMPRSS.COMP.OCSLIB
```

Now you can enjoy the convenience of accessing compressed files. Once decompressed, files remain decompressed until compressed again by LIBRARIAN (unless you use the temporary version.)

Error Conditions

The **FOPEN** trap and subsequent decompression are designed to be completely transparent, although you may notice a slight delay when accessing large compressed files. In the event that an error occurs during decompression, check a JCW called OCSERR to reference the error number.

- Errors *less* than 1000 are file system errors encountered during decompression.
- Errors *greater* than 1000 are process creation errors (subtract 1000 to obtain the true **CREATEPROCESS** error).

For example, error 1001 is **CREATEPROCESS** error number 1 which translates to "Caller lacks Process Handling (PH) capability".

If your programs check condition codes and call FCHECK, a file system error of 560 will be returned for file system errors during decompression, and 561 will be returned for process creation errors. If FERRMSG is called, the following error messages are returned:

```
UNABLE TO DECOMPRESS FILE FOR ACCESS. (FSERR ###)

ERROR CREATING PROCESS TO DECOMPRESS FILE. (CREATEPROCESS
ERR  ###)
```

If your application does not use **FCHECK** and **FERRMSG**, check the OCSERR JCW if the program fails to access a compressed file. This will help you determine the problem.

Appendix B
LIBRARIAN Utility Program

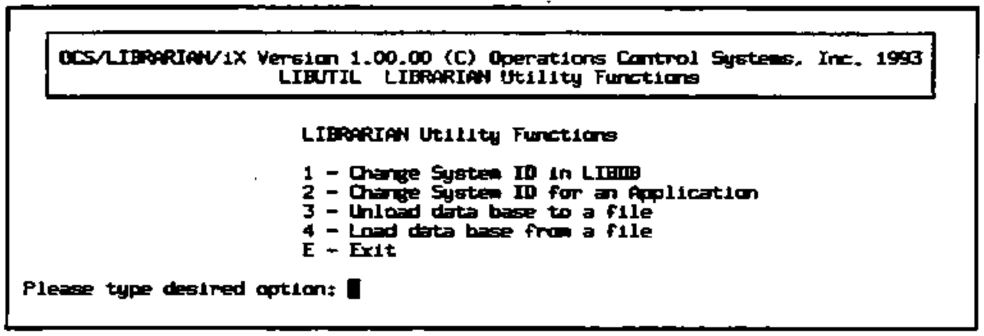
A utility is available for miscellaneous functions including globally changing system IDs. This utility facilitates moving applications or an entire LIBRARIAN implementation to a new system.

Operation

The LIBRARIAN Utility program is called **LIBUTILP.COMP.OCSLIB**. To use **LIBUTILP**, log on as MGR.OCSLIB on the MPE server and type:

```
:RUN LIBUTILP.COMP.OCSLIB
```

The program presents a menu of options as displayed in Figure B–1.



```
OCS/LIBRARIAN/iX Version 1.00.00 (C) Operations Control Systems, Inc. 1993
                   LIBUTIL  LIBRARIAN Utility Functions

                        LIBRARIAN Utility Functions

                        1 - Change System ID in LIBDB
                        2 - Change System ID for an Application
                        3 - Unload data base to a file
                        4 - Load data base from a file
                        E - Exit

Please type desired option: █
```

Figure B–1. LIBRARIAN Utility Functions Menu

This appendix describes the options on the utility menu.

Changing the System ID in LIBDB (Option 1)

You can use **LIBUTILP** to copy your LIBRARIAN database to a file, scan and replace occurrences of the system ID, erase the database, and then reload it. We strongly suggest that you make a backup copy of LIBDB before executing this function.

After you have accessed the Utility Functions menu, enter **1** to select the "Change System ID in LIBDB" option. You will see the following prompts (sample responses are included):

```
Do you have a current backup of LIBDB? (N/Y)  Y

Old System ID: VENUS
New System ID: MARS
```

After you respond to the prompts above, you will see messages similar to the following:

```
Changing System ID from "VENUS" to "MARS".

Unloading LIBDB ..............
Erasing LIBDB ..................
Reloading LIBDB ..............
Entries changed: 78
```

Changing the System ID for an Application (Option 2)

You can use **LIBUTILP** to change your system ID for a specific application.

After you have accessed the Utility Functions menu, enter **2** to select the "Change System ID for an Application" option. You will see the following prompts (sample responses are included):

```
Do you have a current backup of LIBDB? (N/Y)  Y

Application: LOUS

Old System ID:  MARS
New System ID: ( VENUS )
```

After you respond to the prompts above, you will see messages similar to the following:

```
Changing System ID from "MARS" to "VENUS" for Application "LOUS".

Unloading LIBDB..............
Erasing LIBDB ..................
Reloading LIBDB..............
Entries changed: 78
```

Unload Database to a File (Option 3)

You can use **LIBUTILP** to unload the database to a file.

After you have accessed the Utility Functions menu, enter **3** to select the "Unload data base to a file" option. You will see the following prompt:

```
Unload data base (LIBDB):
Comment : THIS IS A LIBDB UNLOAD TO BINARY FILE.

Unloading LIBDB data base to file LIBDBUL. Automatic masters are
ignored.
```

Note The "Unload data base" prompt requests the name of the database root file. The unload file will be the name of the root file with a suffix of "UL".

After you respond to the prompt above, you will see the following message:

```
LIBDB unloading complete.
```

Load Database from a File (Option 4)

You can use **LIBUTILP** to load the database from a file.

After you have accessed the Utility Functions menu, enter **4** to select the "Load data base from a file" option. You will see the following prompt:

```
Load data base (LIBDB):

Loading LIBDB data base from LIBDBUL file

LIBDB data base unloaded on WED, JU. 3, 1992. 9:41 AM
THIS IS A LIBDB UNLOAD TO BINARY FILE.

Loading data....loading M-USER , 1 entry loaded WED, JU. 3, 1992. 9:42
AM
```

After you respond to the prompt above, you will see the following message:

```
LIBDB load complete.
```

Exiting the LIBRARIAN Utility Program

To exit the **LIBUTILP** program, type **E** at the Utility Functions menu.

Appendix C
LIBRARIAN Configuration Program

A utility is available for updating your LIBRARIAN configuration, changing database passwords, and changing server passwords on client systems. Topics in this appendix include:

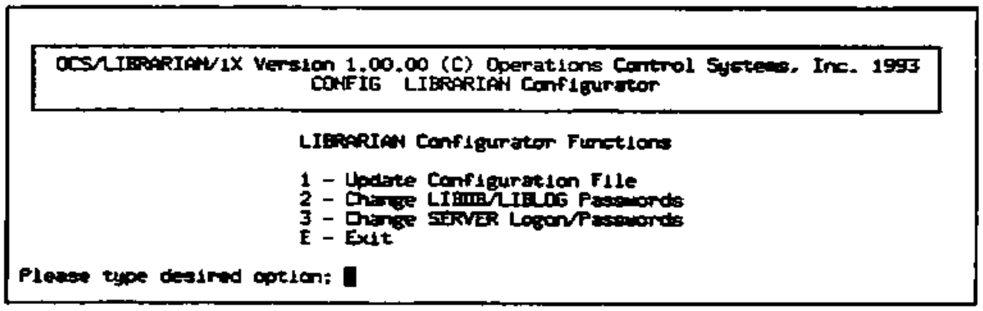
- Configuration Program (MPE)
- Configuration Program (UNIX)

Configuration Program (MPE)

The LIBRARIAN Configuration program is called **CONFIGP.COMP.OCSLIB**. To use **CONFIGP**, log on as MGR.OCSLIB and type:

```
:RUN CONFIGP.COMP.OCSLIB
```

The program presents a menu of options as displayed in Figure C-1.



```
  OCS/LIBRARIAN/iX Version 1.00.00 (C) Operations Control Systems, Inc. 1993
                       CONFIG  LIBRARIAN Configurator

                       LIBRARIAN Configurator Functions

                       1 - Update Configuration File
                       2 - Change LIBDB/LIBLOG Passwords
                       3 - Change SERVER Logon/Passwords
                       E - Exit

Please type desired option: █
```

Figure C-1. LIBRARIAN Configuration Functions Menu

Updating the Configuration File (Option 1)

Enter **1** to select the "Update Configuration File" option. You will see the following prompts:

Warning Do not attempt to update your configuration file if anyone is accessing LIBRARIAN.

```
Change Configuration Values (Use // to Remove Value)

Current COMPANY NAME    : OCS
  New COMPANY NAME      :

Current SYSTEM ID       : VENUS
  New SYSTEM ID         :

Current EDITOR          : EDITOR.PUB.SYS
  New EDITOR            :

Current HOST ID         :
  New HOST ID           :

Current BATCH LOGON     :
  New BATCH LOGON       :
```

After you respond to the prompts above, you will see the following message:

```
Configuration updated successfully!
```

Changing LIBDB/LIBLOG Passwords (Option 2)

To change passwords in the LIBDB and LIBLOG databases, enter **2** to select the Change LIBDB/LIBLOG passwords option. You will see the following prompts:

```
Current LIBDB READ password     :
  New LIBDB READ password       : TOP

Current LIBDB WRITE password    :
  New LIBDB WRITE password      : SECRET

Current LIBLOG READ password    :
  New LIBLOG READ password      : FORYOUR

Current LIBLOG WRITE password   :
  New LIBLOG WRITE password     : EYESONLY
```

After you respond to the prompts above, you will see the following message:

```
Changing database passwords...done.
```

Changing SERVER Logon/Passwords (Option 3)

To change the LIBRARIAN server logon and passwords in the LIBDB database, enter **3** to select the Change SERVER Logon/Passwords option. You will see the following prompts:

```
Current USER              : CLIENT
  New USER:

Current USER password     :
  New USER password       :

Current ACCOUNT           : LIXTR
  New ACCOUNT             :

Current ACCOUNT password  :
  New ACCOUNT password    :

Current GROUP             : PUB
  New GROUP               :

Current GROUP password    :
  New GROUP password      :
```

To allow HIPRI login, type HIPRI; otherwise type //

```
Current HIPRI             :
  New HIPRI               : //
```

After you respond to the prompts above, you will see the following message:

```
Configuration updated successfully!
```

Note In order to use **HIPRI** for automatic remote logon to LIBRARIAN, MPE requires that the logon user and account have OP or SM capability.

Exiting the LIBRARIAN Utility Program

To exit the **CONFIGP** program, type **E** at the Configuration Functions menu. You return to the main LIBRARIAN prompt.

Configuration Program (UNIX)

If you are running a UNIX client, you can configure information about the MPE server, including passwords for gaining access. In addition, you can update the ocslib user password that already exists on the client.

You must have superuser capability to run the configuration program. The following is an example:

```
HP-UX [1] cd /opt/ocs/ocslib
HP-UX [2] su
Password:
# ./ocslib -config

Change Server Login/Passwords (^X  to remove password)

SERVER NAME [PENGUIN]:
USER [CLIENT]:
USER PASSWORD (not displayed):
PLEASE RE-ENTER PASSWORD TO VERIFY:
ACCOUNT [ocslib]:
ACCOUNT PASSWORD (not displayed):
PLEASE RE-ENTER PASSWORD TO VERIFY:
GROUP [PUB]:
GROUP PASSOWRD (not displayed):
PLEASE RE-ENTER PASSWORD TO VERIFY:
UNIX LOGIN [ocslib]:
UNIX PASSWORD (not displayed):
PLEASE RE-ENTER PASSWORD TO VERIFY:
Configuration /opt/ocs/ocslib/config updated successfully!
```

LIBRARIAN/iX Glossary of Terms

Note Terms that appear in *italics* in the following definitions have separate glossary entries.

A

Access Control

The attribute of a *master file* that determines how many *read/write mode* copies are allowed. The four access control levels are: *exclusive, read only, serial write, and multiwrite.*

Access Mode

The attribute of a *secondary file* that determines whether or not it can be checked in and replace its associated *master file*. A secondary in *write mode* can replace a master. A *read mode* can only replace a master through an *emergency checkin* that is configured to use the *PUSHREAD* parameter. A file's access mode is determined by *access control*, user request, *step* definition, and *default access mode* (precedence is in order listed).

Aging Policy

A *system profile* value that indicates how long log records are kept. When the *FLUSHLOG* utility is run, audit trail records that are older than the number of days specified in the aging policy are deleted.

Transactions associated with projects override this policy and are deleted only when the project status is flush pending.

Alternate prestep

A *prestep* that can be performed as an alternative to the defined prestep. Up to three alternatives can be defined for a *step*.

Annotate

Comments inserted by LIBRARIAN into source listings that indicate which lines were inserted/deleted for which *revision*. Date/time, related project and user who made the change are included.

Application

A site–defined organizational unit including a set of *master files* that are being controlled by LIBRARIAN, a set of *steps* for file movement/approval, and, optionally, a set of *projects* for tracking file changes associated with a particular work activity.

Application Manager

A *special user capability* assigned to the user responsible for the files and *steps* within an *application*.

Application fileset

The highest level *fileset* for an *application.*

Approval step

A *null step* that is required as a prerequisite for a subsequent step.

Authorization

The process of determining which files have been requested in a *transaction* and whether or not the rules permit the operation to be performed on each of these files. Authorization is based on the user who initiated the request and the current status of each file requested.

AUTOXEQ file

A *macro* that is executed before the first prompt/main menu appears. A file called AUTOXEQ that exists in the product account is executed prior to any AUTOXEQ file that might exist in the user's home directory.

Auto fileset descriptors

General locations that describe how *master files* are assigned automatically to *master filesets.* Descriptors can include or exclude files from filesets using *wildcards.* When you run *AUTOUPDATE,* introduce new files with a *pending master,* or perform a *checkin step* with the AUTOUPDATE parameter turned on, any previously *untracked files* in these locations get added to the appropriate master filesets.

Automatic Login ID

The login used when transactions require automatic logging in to a remote system.

Autoupdate

The process used to add *master files* to *master filesets* automatically based on predefined *auto fileset descriptors* that include or exclude files from filesets, typically using wildcards. *Pending masters* and masters not currently assigned to required filesets are added, typically during *checkin, new steps* and/or running of the AUTOUPDATE utility.

B

Baseline

The *master library* at a particular point in time. An *application manager* establishes a baseline by creating a *version.* This marks and protects all of the files in an application at that time, so that the application or any part of the application can be restored to that baseline any time in the future.

Base Revision

A *revision* that was current at the time a *baseline version* was created. The *version count (VCOUNT)* for a base revision is always zero and cannot be flushed until the version(s) of which it is a part is made *obsolete.*

Branch

A set of *revisions* that are made as a divergence from the main development path for a master file. A branch is created automatically when a previous revision is checked out. A branch can also be forced from the latest revision if the master is already checked out in *write mode,* or the user does not intend to check the file back in on the *trunk.* Whenever a new branch is created, a branch counter and *leaf* counter (both starting at 1) are appended as a pair to the original *revision ID.*

Branch revision

A *revision* that appears on a *branch.*

C

Checkin step

Any *step* which copies or moves a file from a *secondary location* into the *master library,* either retaining and replacing the existing master, introducing a new one or establishing a new branch .

Checkout step

Any *step* which copies a file from the *master library* into a *secondary location,* generally for modification by programmers.

Client

An MPE or UNIX implementation of LIBRARIAN where the LIBRARIAN data bases reside on a different system, but the user is able to perform all LIBRARIAN functions.

Command Mode

In command mode, the user enters LIBRARIAN commands at a command line prompt. Users can switch between command mode and *menu mode* by pressing the F2 function key.

Component filesets

Filesets that are subsets of higher–level filesets.

Composite prestep

A collection of *presteps* that must be performed before a subsequent step can be performed. Composite presteps also permit the specification of a date prerequisite.

D

Default access mode

The *access mode* that is assigned to a *secondary file* when neither the user or *step* explicitly specify the mode. The *access control level* for a file determines which access modes are allowed.

Delta file

A privileged (MPE) or hidden (UNIX) file that contains the history of changes made to an associated *master file.*

Deltas

A method for retaining and reconstructing previous revisions of *master files* that involves storing only the changes to files over time.

Dependency

A file that *make* evaluates with respect to some target to determine whether to invoke some action, such as a compile or link.

Destination

The target location when copying or moving a file.

Dummy target

A *make target* that does not correspond to an actual file. *Dependencies* of dummy targets are actual files that are always evaluated as targets themselves to determine whether they are out of date and need to be rebuilt.

E

Edit mask

A file expression that uses special editing characters to map one filename into another; e.g., source to destination name for a copy or move or *secondary* to *pending master name* for introduction of a new file.

Emergency checkin

A checkin that moves a *read mode secondary file* into the *library* with the *PUSHREAD* option. If a *write mode* copy exists, the *owner* is notified via a LIBRARIAN *mail* message, and an *exception* is recorded.

Exception Flag

An indicator that something special has happened related to a file such as an *emergency checkin, merge conflict* or previous *master revision* was restored at a time when the file was checked out. The exception flag must be cleared before any further operation on the file is allowed.

Exception message

A LIBRARIAN *mail* message that indicates that an exception flag has been placed on a file. This message is sent to the *owner* of the *write mode* copy of the file.

Exclusive access

The *access control level* that prevents *secondary* copies of a *master file* from being made.

Expiration date

The date when after which a file can be flushed using the *FLUSH* utility.

Expired file

A *read mode secondary* or *retained file* that is eligible to be flushed by the *FLUSH* utility.

Explosion

The creation of a list of files by expanding a *fileset, listfile,* or *wildcard* file specification for LIBRARIAN to *authorize.*

External

A file that resides on a system on which LIBRARIAN is not running, typically an unsupported platform, or system which is not on an accessible network. LIBRARIAN steps can be used to record movement to an external location, but cannot physically move the file or verify its existence. Users are responsible for transferring files (via tape or other means) for any transaction using the EXTERNAL option.

F

Fileset

A collection of files identified by a unique name assigned by the *Librarian Manager* (*master filesets*) or any user (*user filesets*). When requesting files, filesets can be referenced by preceding the fileset name with a percent sign (%). Because filesets contain collections of files that are related by some criteria other than physical location, and can span directories and systems, they are often referred to as *logical filesets.*

Note: In MPE, a fileset is any set of files that can be referred to using wildcards in name, group and/or account. LIBRARIAN refers to this as a *physical fileset.*

File structure (hierarchy)

The relationship of filesets, subsets and physical files within an application library.

Flush policy

The *system profile* policy that determines how many previous file *generations* to keep when the *FLUSH* maintenance utility is run.

FLUSHLOG

The maintenance utility that purges old log records that have aged beyond the *aging policy* specified in the *system profile.*

FLUSH

The maintenance utility that purges *expired files* and *obsolete versions.*

Flushed project

When a project is closed and then assigned a status of flush pending, log records associated with that project get flushed the next time the *FLUSHLOG* utility is run. After FLUSHLOG has been run, the project status is changed to flush, and the project can be deleted, if desired.

Flushed version

When a *version's* status has been changed to *obsolete,* base *revision* files that are a part of that version are flushed if they are not also part of a subsequent version. After *FLUSH* has been run, the version status is changed to flush, and the version can be deleted, if desired.

Flush pending

A *project status* that indicates that log records for the *project* should be purged when the *FLUSHLOG* utility is run.

FMAINT

The facility for creating and maintaining *user filesets.*

Forward versioning

An option on *checkout* to automatically search alternate *libraries* (usually previous versions) when a *master file* is not found in the expected *location* as defined by the checkout step. If the file is then found in an alternate location, it is brought forward as a *secondary* of a new *pending master* for the primary *application.*

G

Generation

Each time a file is checked in, a new generation is created. Previous generations of *master files* are stored in the *library* as *retained files* (usually compressed) or as *deltas.*

Generation count (GCOUNT)

A sequential number assigned to each *master file generation.* The current GCOUNT is the total number of times a master file has been replaced. When specifying GCOUNT as an option in a file request, a negative number indicates a generation relative to the latest generation.

Generic rule

A *target–dependency* relationship in *make* that uses *wildcards* (target) and edit masks (dependency) to determine what is out of date. Actual target and dependency names are substituted into the rebuild commands using *make macros.*

I

Indirect file

Also called a *listfile,* an indirect file is a text file that includes a list of filenames. This file can be used in *LIBRARIAN* commands as a convenient way of referencing files. Indirect files can be created in a text editor or through *LIBRARIAN's LMAINT* facility.

INPROGRESS

A parameter used with a *checkout step* that instructs LIBRARIAN to record the existence of a *write mode secondary* without physically copying the file from the *library.* This parameter is most often used when LIBRARIAN is initially implemented and some files are already being worked on or tested.

Intermediate revision

Master files that are retained between versions. The version count (VCOUNT) for intermediate revisions is always greater than 0.

L

Leaf Revision

Each *revision* on a *branch* is called a leaf, sequentially numbered from the start of the branch. Whenever a new branch is created, a branch counter and leaf counter (both starting at 1) are appended as a pair to the original *revision ID.*

LIBRARIAN

The program that controls and processes all file operations maintaining an audit trail of activity.

LIBRARIAN Manager

A *special user capability* assigned to the person responsible for configuring LIBRARIAN and defining site rules. The LIBRARIAN Manager has unrestricted access to all LIBRARIAN functions for all files.

Library

A library is the repository from which files are *checked out,* and to which they are subsequently *checked in.* Files are also distributed to production locations from the library. It is the 'official' collection of files that are under LIBRARIAN's control. Files in the library are called *master files.* The library provides a central point of control for changes to production source, object and data.

Listfiles

Also called an *indirect file,* a listfile is a text file that includes a list of filenames. This file can be used in *LIBRARIAN* commands as a convenient way of referencing files. Listfiles can be created in a text editor or through *LIBRARIAN's LMAINT* facility.

LMAINT

The facility for creating and maintaining *listfiles (indirect files).*

Location

The group/account (MPE) or directory (UNIX) and *system* where a file exists or should be created.

Logical fileset

A meaningful name assigned to a collection of files not bound by physical boundaries. See *fileset*.

!LOGON, !LOGIN

A special wildcard that can be used in defining step source and destination *locations* to indicate that the user's login data should be substituted as appropriate. For MPE, this wildcard can be used for group, account and/or *system*. For UNIX, this wildcard is equivalent to '.' for current working directory and can also be used for *system*.

M

Macro

A set of *LIBRARIAN* and operating system commands for LIBRARIAN to execute. A macro *control language* provides programmatic control (conditions and loops) and parameter substitution. Parameter values can be system–defined or provided by the user via prompts and/or customized menus. Macros are analogous to MPE command files and UNIX scripts. Multiple macros can be combined in a single *procedure file*. Macros are also referred to as *XEQ files*.

Macro Control Language

The set of special commands and keywords that are used in macros to control flow of execution (IF...THEN...ELSE, REPEAT, WHILE, LOOP, GOTO) and allow for parameter substitution (tokens preceded by %%).

Mail

Mail includes messages that are sent from one LIBRARIAN user to another, or from *LIBRARIAN* notifying a user that an *exception* condition has occurred that affects that user's work.

Make

A utility that automatically rebuilds/recompiles components of an *application* when they change. Make reads a *makefile* that shows *dependencies* between application components and evaluates which components are out of date. Based on which components are out of date, make issues only the commands necessary to bring the application up to date.

Makefile

A text file that contains make rules. This file can have any name and can be created and maintained using any text editor. This file includes *target–dependency* relationships and commands required to bring each target up to date whenever their dependencies are changed. *Make macros* and *generic rules* can be used to reduce the size and complexity of a makefile.

Make macros

A shorthand that simplifies creating *makefiles*. Macro references are substituted with either user–defined or system–defined values when the makefile is processed. For example, out–of–date *dependency* names can be substituted in generic command descriptions.

Master file

A file that is part of a defined *library* and reflects the most current production version.

Master fileset

A *fileset* defined by the LIBRARIAN Manager that includes *library* files.

Master library

The hierarchy of *master filesets* and associated *master files* for an *application*.

Memo

Text that provides documentation for a *transaction*. Memos are stored in the audit trail database and can be reviewed using *SHOWLOG*.

Menu Mode

The mode of *LIBRARIAN* operation in which users select LIBRARIAN functions from a set of pull–down menus. Users can switch to the command line prompt at any time by pressing the F2 function key.

Merge

An option available on *checkout steps* to combine source code changes from one or more *branches*. Conflicting changes are highlighted with comments in the source code, and should be resolved prior to the next step. Merge is only available if the *delta* feature is being used.

!MSUSER

A special *wildcard* that can be used in defining step destination *locations*. When the step is executed, the wildcard is replaced with the user ID of the user who originally checked out the file. For MPE, this wildcard can be used to fill in group or account. For UNIX, this wildcard can appear anywhere in the path name. This wildcard is typically used to reject files and move them from a test area back to the appropriate developer's work area.

Multi–write

The *access control* level that allows multiple *secondary files* with *write–mode* access.

N

New step

A *step* that introduces a previously *untracked file* to *LIBRARIAN* as a *secondary file*. The file is linked to a pre–existing *master file* or a *pending master* record is created. Rules governing introduction of new files on a step are configured on the PP (Pending Production Areas) screen.

Node

The actual device name associated with a system in a network. This name may or may not be the same as the LIBRARIAN *system ID*.

Null step

A *step* not involving any file movement. A null step is used to reflect some external action such as an approval. Null steps are used to control dependencies between steps; that is, they are used as *presteps*.

O

Obsolete version

When the *LIBRARIAN Manager* or *Application Manager* change the status of a *version* to obsolete, any *retained base revisions* associated with that version will be flushed the next time the *FLUSH* utility is run. Once a version is flushed, it can be deleted, if desired.

Operator

A *special capability* assigned to a user who can *flush* records in the log database and can restore previous revisions of files.

Orphan

Any file not currently being tracked by *LIBRARIAN* or a *master file* not associated with an *application*. Orphans can be created by a LIBRARIAN operation that causes a tracked file to become untracked (unknown to LIBRARIAN), or by operations that use the orphan option to create files in destinations that are not to be tracked.

!OWNER

A special *wildcard* that can be used in defining step destination *locations*. When the step is executed, the wildcard is replaced with the user ID of the user who currently owns the file. For MPE, this wildcard can be used to fill in group or account. For UNIX, this wildcard can appear anywhere in the path name. This wildcard is typically used to approve files in multiple developer work areas.

P

Parent Fileset

A *fileset* that includes *component* filesets.

Pending master file

A file that is being *tracked* as a *master library file*, but, because it is new, does not physically exist in the *library* yet. The associated *secondary* is called a *pending production file* and was introduced through a *new step* or through the use of LIBRARIAN's *forward versioning* feature.

Pending master mask

An *edit mask* used to automatically derive a *pending master file* name based on the name of the *secondary file* being introduced through a *new step*.

Pending production area

Any *location*(s) defined for a *step* where previously *untracked files* can be introduced as new *secondary files*. Steps with pending production areas are considered to be *new steps*.

Pending production file

A *secondary file* that was introduced using a *new step*. The *master file* does not currently exist in the *library*.

Permissions

A UNIX term used to indicate file access rights; a matrix of read, write, and execute access for owner, group and world.

Physical fileset

A collection of files that exist in a particular *location*. Physical fileset references include specific filenames, or names using standard operating system/shell *wildcards*.

Prestep

A *step* that must be completed successfully for a file before the next step in the *route* can be performed. Presteps are often *null approval steps*.

Procedure

A *macro* that is included in a file with other macros with a procedure header.

Procedure file

A file that contains multiple *macros*. Each macro has a procedure header indicating the name of the macro. Procedure files can be loaded and unloaded while using *LIBRARIAN*.

Project

A way of organizing *transactions* and associated files with a specific work activity.

Project fileset

A *user fileset* that is created automatically when defining a *project*. The fileset is maintained automatically when files are *checked out* or introduced as new files for the project. Files can also be added to this fileset in advance by a *Project Manager* using the *FMAINT* facility.

Project manager

A *special user capability* assigned to users who can create projects, modify project status and authorize users to work on projects.

Project menu

Whenever *projects* are associated with a particular *route*, users are asked to select the project that they are working on from a menu when checking files out or introducing new files.

Project status

A flag that determines what activities can be associated with a *project*.

PUSHREAD

A *step* option which allows a *read mode* copy to replace a *master file* or *write mode secondary* which has not been checked in yet. This option is typically used for *emergency steps*.

R

Read mode

The attribute of a *secondary file* that indicates it cannot replace the *master*. Read mode copies expire after a configured period of time and can be flushed using the *FLUSH* utility.

Read only

An *access control* level that only allows *read mode* copies of a file.

Read step

A *step* that copies a *master file* to a *secondary* location in *read mode*, with no intention for modification. An *expiration* policy can be applied, so that read mode copies created by the step can be cleaned up automatically with the *FLUSH* utility.

Receiver

A *system* that can receive files from other systems, but from which *LIBRARIAN transactions* cannot be initiated.

Release Step

Similar to a *read step*, a release step copies files from the *library* to a production *location* in *read mode*. Typically, these files do not expire, and the previous version is often *retained*.

Retained file

A previous *generation* of a file saved under a *LIBRARIAN*–generated name "G#######". Files are retained when the retain parameter is used on a *step* and the destination file is a *tracked master* or *secondary file*. *Base revisions* are always retained. If *deltas* are being used, changes to the previous generations are stored.

Revision

Any set of changes made to a *master file* through a *checkin* step. Revisions include all *generations* of a master file including the most current. *Leaves* and *branches* also make up the set of revisions for a file.

Revision ID

Revisions are identified by version name followed by a colon (:) followed by version count. If the revision is on a branch, branch and leaf count pairs are appended delimited with periods (.).

Route

A set of automated procedural controls for managing file changes and distribution. A route consists of a predefined file–movement path that reflects an established cycle. The route includes *steps* for all allowable movements of the files for that cycle.

Route Alias

When defining *projects*, a route alias can be defined to indicate that the project only applies to a particular *route*. The project name can be used in place of the route name when performing a step (i.e., step.project) to bypass the *project menu*.

Rule Administrator

Similar to the *LIBRARIAN Manager*, the Rule Administrator is a user with *special user capability* who can define *LIBRARIAN* rules such as steps and filesets, but is not automatically authorized to perform LIBRARIAN functions, and cannot create *user authorizations*.

S

Scan/Replace

A *LIBRARIAN* function that searches files for patterns of text, and optionally replaces the matches with user–defined text.

Scope

The attribute of a *step* that restricts which files the user can request. When copying or moving files, the scope specifies where files come from and where they can be copied. Steps can restrict by fileset, from location and to location.

Secondary file

Any copy of a *master file* or another secondary file. All secondaries are linked to a master (or *pending master*) either directly or indirectly, and are in *read or write mode*.

Secondary location

Any *location* where *secondary files* can be created.

Serial write

The *access control* level that allows only one *secondary file* at a time to have *write mode* access, preventing concurrent modifications.

Server

A system that has an implementation of *LIBRARIAN* which includes the LIBRARIAN databases. *Clients* access this database and other LIBRARIAN functions remotely.

Settings

LIBRARIAN session-level parameters that control the user's working environment.

Special user capability

See *user capabilities*.

Standard Rule

A *make* rule that associates specific *target*(s) with specific *dependencies*.

Step

A rule governing the copying and moving of files from one *location* to another. Steps are the basic building blocks of the *LIBRARIAN* file movement and control system. Steps are grouped into *routes* and are performed using system– and/or site–defined names.

Step parameter defaults

Options that control the behavior of a step, by default.

Step parameter overrides

If allowed, users can override *step parameter defaults* by specifying desired overrides.

Step refinements/exceptions

A *step* definition that includes rules for altering the destination *location* based on the from location, filecode (MPE), and/or *fileset* membership. The same criteria can be used to alter the type of movement (copy, move or null) or exclude files altogether from the step.

Step type

There are three types of steps: master–to–secondary (MS), secondary–to–secondary (SS) and secondary–to–master (SM). MS steps are steps that checkout or distribute files. SM steps are steps that check files in. SS steps encompass all steps in between, such as move to test and approvals.

System

A unique *node* within a network identified to *LIBRARIAN* with a unique *system ID*.

System ID

Used to *identify* systems to *LIBRARIAN* within a network. Optionally appears as a prefix to a filename delimited by ':' to indicate the appropriate system.

System Profile

A set of global parameters maintained by the *LIBRARIAN manager* that control how LIBRARIAN operates. Includes items such as flush policy, aging policy, date formats, etc.

T

Tag

A user–defined name for a particular *revision* of a file or files that can be used to identify them at a later time, even after they have been *retained.*

Target

Component of a make rule that is built from one or more dependencies using one or more commands. Object code and executables are examples of targets.

Tracked file

A file for which there is a record in the LIBRARIAN data base. Tracked files are *masters, secondaries* or *retained files* and movement operations are controlled by *LIBRARIAN* rules. All other files are *untracked files.*

Transaction

Any LIBRARIAN operation attempted either successfully or unsuccessfully on a set of files. Except for commands which provide information, all transactions are logged in the LIBRARIAN audit trail.

Trunk revision

A *revision* that is not checked in on a *branch.*

U

Untracked file

A file for which there is no record in the *LIBRARIAN* database. Ad hoc operations on these files conform to normal operating system security. Steps cannot be performed for untracked files.

User authorizations

The mechanism for determining who can do what. Authorizations can be defined for *steps* and *projects. Special user capabilities* can be assigned so that specific authorization is not required in some cases.

User capabilities

Grants users certain privileges that transcend standard *user authorizations*. These include *LIBRARIAN Manager, Application Manager, Project Manager, Operator, Rule Administrator* and *X capability*. If no special capability is assigned, authorization is required for steps, and other commands conform to normal operating system security.

User fileset

A *fileset* created and maintained by a user through the *FMAINT* user fileset module. User filesets allow users to group files for their convenience. Like *master filesets*, precede user filesets with % when referencing them in commands.

!USERID

A special *wildcard* that can be used in defining step source and destination *locations*. When the step is executed, the wildcard is replaced with the user ID of the user performing the step. For MPE, this wildcard can be used to fill in group or account. For UNIX, this wildcard can appear anywhere in the path name. This wildcard is typically used to check out file's into the developer's work area.

User ID

A unique identifier for a LIBRARIAN user that is password protected. Users are prompted for their User ID when initiating the *LIBRARIAN* program.

User password

Used to protect against unauthorized use of the *LIBRARIAN* system. Passwords are required and can be changed by the individual users.

V

Verify

The *LIBRARIAN* facility for reviewing file information on–line or off–line.

Version count (VCOUNT)

The sequential number that tracks the number of *generations* since the current *version* was defined.

Version

All the files in an *application*, as they were at a specific point in time.

Version ID

The name given to a version by a *LIBRARIAN* or *Application Manager.*

W

Wildcards

Special characters or tokens used in filenames to request multiple files that match a pattern, and/or to determine destination *locations.*

Work-in-progress

Untracked files that were in development and/or test prior to LIBRARIAN implementation. These files can be handled using the INPROGRESS parameter with a checkout step.

Write mode

The attribute of a *secondary file* indicating that it can replace its *master file* through an authorized *checkin step*.

X

XEQ file

A text file that contains the commands for a single *macro*. These macros are executed by filename.

Index