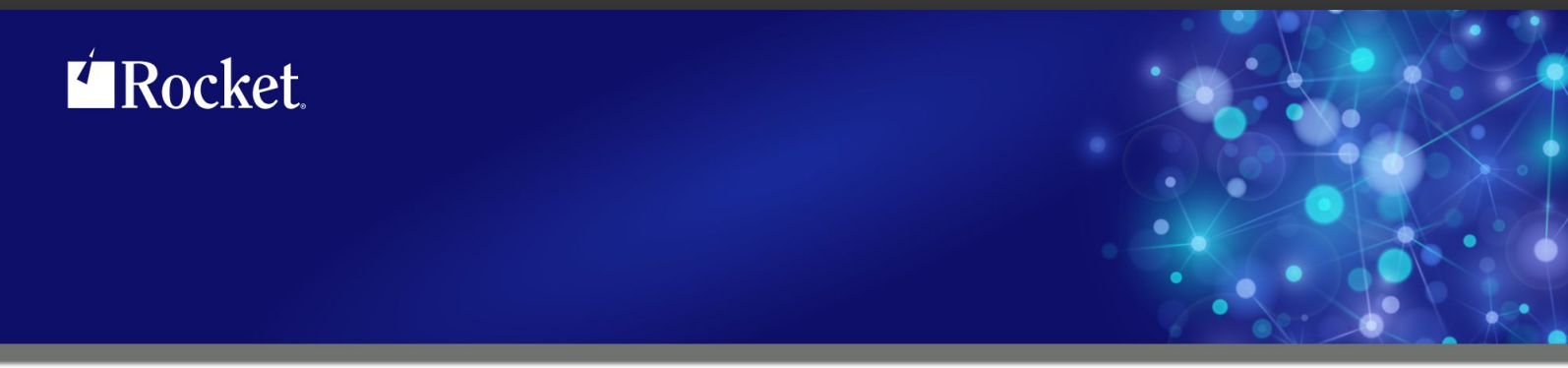


Rocket Model 204 SirLib

User's Guide

July 2013
LIB-0704-UG-01

Notices

Edition

Publication date: July 2013

Book number: LIB-0704-UG-01

Product version: Rocket Model 204 SirLib

Copyright

Trademarks

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

Note

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Contact information

Website: www.rocketsoftware.com

Rocket Software, Inc. Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451–1468
USA
Tel: +1 781 577 4321
Fax: +1 617 630 7100

Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Global Technical Support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone:

North America	+1 800 755 4222
United Kingdom/Europe	+44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

Contents

CHAPTER 1 Introduction to UL/SPF

UL/SPF is a family of products implemented as User Language application subsystems and designed to function together in a *Model 204* Online. Each product in the family can be installed and run independently, or each can be installed and run as a component of the integrated *UL/SPF* (User Language / System Productivity Facilities) framework.

All *UL/SPF* products share a common "look and feel" that is modelled upon IBM's ISPF facilities. *UL/SPF* supplants the sometimes arcane command language of *Model 204*, providing a highly productive full screen interface to a variety of common *Model 204* functions. *UL/SPF* enhances the capabilities of *Model 204* and bypasses many of its restrictions, enabling the performance of routine tasks that were previously impossible or prohibitively time-consuming.

UL/SPF comprises the following products:

SirDBA A system that analyzes *Model 204* databases to determine their logical structure, populating an internal catalog. *SirDBA* is distributed as a component of the *Sir2000 Database Analysis Tools*.

SirFile A comprehensive facility both for monitoring the physical storage utilization of *Model 204* database files and for warning users of the need for file reorganizations. *SirFile* maintains historical information that allows it to predict when file sizing problems *will* occur, allowing a DBA to take preventative action before an application outage results.

SirLib A system that provides change management and configuration control for *Model 204* User Language applications. Fully integrated within the programming environment, *SirLib* supports unique *Model 204* constructs such as file groups, while remaining nearly transparent to programmers.

SirMon A comprehensive facility for monitoring the performance and availability of *Model 204* online systems. *SirMon* combines the real time monitoring of *Model 204* performance with intelligent full screen displays that facilitate System Manager duties.

SirPro A collection of powerful and easy to use tools for programmers, database administrators, and application managers. *SirPro* provides programmers with powerful facilities for managing large libraries of User Language procedures, and it provides system managers with intuitive ISPF-like front ends to many *Model 204* system management commands.

SirScan A high performance utility that allows users in a *Model 204* Online to browse the contents of its journal in real time. *SirScan* permits ordinary

users to view journal entries generated by their own online session, and it allows users in ADMIN SCLASSes to browse journal entries for any set of users. The data is displayed in a full-screen browser with powerful searching commands and filtering options.

In addition, a number of subsystems that are not linked into the *UL/SPF* menuing structure may be accessed via APSY-transfer from the *UL/SPF* applications. One such subsystem is **FACT**, a utility for browsing *SirFact* dumps.

There are also many sample web and client-server applications developed for the Janus product family that are distributed and installed along with the *UL/SPF* products. These include **JANCAT**, an application that builds normalized views of *Model 204* data for use by *Janus Specialty Data Store* applications, and **JANSSL**, a system for creating and managing SSL certificate requests.

1.1 UL/SPF packaging and installation requirements

All of the *UL/SPF* products are *Model 204* application subsystems written in User Language. *UL/SPF* is distributed as a set of *Model 204* files in a backup format produced by the *Model 204* DUMP command. All User Language based products are distributed in a single *Model 204* procedure file called **SIRIUS**.

UL/SPF makes extensive use of specialized User Language $functions that enable the creation of User Language application systems that can support complex environments with minimal server size requirements. Prior to version 7.5 of Model 204, the $functions were part of the *Sirius Mods*, the installation of which (*Sirius Mods Installation Guide*) was a prerequisite for any *UL/SPF* product.

The *UL/SPF* $functions are included in the **http://m204wiki.rocketsoftware.com/index.php/List_of_$functions**, and their use is controlled by a product authorization mechanism. Many functions require that a specific *UL/SPF* product be authorized. Some functions may only be invoked from an authorized procedure “signed” by Sirius Software or Rocket Software. The `SIRIUS` command examines the status of all Sirius products installed in a *Model 204* Online.

1.2 Integrating UL/SPF with other subsystems

UL/SPF and any of its constituent products can be easily integrated with other User Language subsystems. Whenever a *UL/SPF* component product is exiting, it first checks to see if the global variable `SIRIUS.COMM` exists and has a non-null value. If so, the *UL/SPF* product performs a subsystem transfer using the value in SIRIUS.COMM as the name of the target subsystem.

For example, the following code fragment lets you transfer into SIRMON. When SIRMON exits, control is transferred to the application subsystem MENUSYS, provided that NEXTPROC is the current subsystem's communication global variable:

```
%RC = $SETG('SIRIUS.COMM','MENUSYS')
%RC = $SETG('NEXTPROC','XFER')
%RC = $SETG('XFER','SIRMON')
STOP
```

In addition to the individual User Language subsystems that implement the *UL/SPF* products, an umbrella *UL/SPF* subsystem, **ULSPF**, provides a menu that contains entries for all *UL/SPF* components installed at a site.

The *UL/SPF* subsystems support fast path navigation. For example, a *SirMon* user can transfer into *SirPro* Option 1 by typing:

```
=M.1.1
```

1.3 Related documentation

The following documentation is available from **http://docs.rocketsoftware.com** (M204 folder) or from **http://m204wiki.rocketsoftware.com**:

- ***SirDBA User's Guide***
- ***SirFile User's Guide***
- ***SirLib User's Guide***
- ***SirMon User's Guide***
- ***SirPro User's Guide***
- ***SirScan User's Guide***
- ***Sirius Mods Installation Guide***
- ***Model 204 System Manager's Guide***

The following documentation is available from the Model 204 documentation wiki (**http://m204wiki.rocketsoftware.com**):

- UL/SPF installation (http://m204wiki.rocketsoftware.com/index.php/UL/SPF_installation_guide)
- $functions (http://m204wiki.rocketsoftware.com/index.php/M204wiki_main_page#.24Functions)
- Model 204 commands (http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_commands)
- Model 204 files (http://m204wiki.rocketsoftware.com/index.php/Category:File_architecture_and_management)
- UL/SPF product messages (http://m204wiki.rocketsoftware.com/index.php/Category:Sirius_Mods_messages)
- User Language/SOUL (http://m204wiki.rocketsoftware.com/index.php/Category:SOUL)

CHAPTER 2 Overview of Configuration Management

After reading this chapter, you should read "Getting Started" on page 61 for tips on configuring your *SirLib* and *SirPro* environment.

SirLib runs as an APSY subsystem inside a *Model 204* online, as does *SirPro*, a package of productivity tools. Together they provide a complete, integrated User Language developers' environment. *SirPro* provides the programmers' environment, and *SirLib* provides the management environment for applying and backing out updates.

There is some overlap in programmer and management activity in managed environments, and there is no reason why programmers cannot be responsible for performing *SirLib* functions. This document covers briefly the *SirPro* functions related to managed changes. These functions are also covered in the ***SirPro User's Guide***.

Configuration management functions typically fall into four categories:

- Identifying, grouping, and documenting system changes.
- Providing an environment for controlling change and programmer activity.
- Backing out changes.
- Reporting on system configurations.

SirLib provides specific functions for each facet of change control. Furthermore, *SirLib* reduces the complexities of procedure distribution to such an extent that only a single, small procedure file needs to be distributed to production environment(s), no matter how many applications run in that environment.

In addition, *SirLib* is designed to provide specific mechanisms to automate and simplify programmers' tasks:

- *SirLib* prevents programmers overwriting each others' changes. In typical change management systems, this involves a procedure lock. *SirLib* performs this function without locking procedures.

- *SirLib* allows developers to continue working in the manner they're used to. *SirLib* does not dictate a development method.

- *SirLib* allows for quick fixes and fast backout of changes when introduced changes cause production problems.

- Reporting capabilities are integrated, and the data for reporting does not have to be manually entered — reporting data is generated to the internal dataset **SIRLIBD** when system activity occurs.

- Verification of change level (or status, release number, etc.) is available in all environments where *SirLib* runs, via both reports and optional internal procedure comments.

Most change management systems will not work in *Model 204*, because:

- They cannot be integrated with developer tools.
- They cannot handle the long procedure names permitted in *Model 204*.
- They don't understand the concept of *Model 204* GROUPs.

SirLib was designed to deal with all these change management issues and to work under a wide range of *Model 204* configurations. *SirLib* provides primarily a *means* to achieve a well-managed User Language environment: it doesn't dictate procedure promotion schemes, and it doesn't force any project to manage its changes the same way as other projects in the same shop. A shop can standardize on a single mechanism, while allowing each project to promote changes via a path that is as simple and straightforward as possible.

In addition, *SirLib* allows multiple users to make updates to the same procedure at the same time, and it eliminates the risk of programmers overwriting each others' changes. Essentially the granularity of an update is reduced to a *section* of a program, and possible update collisions are detected and handled unambiguously and early in the development process.

A *managed update* refers to any change to a *Model 204* procedure which is managed by the change management system. In this document, that means any change generated using *SirPro* and applied using *SirLib*. Managed updates may be small bug fixes, major application enhancements, or additions and deletions of whole procedures. Managed and unmanaged updates are made via the full screen interface to the *Model 204* editor, *SirPro*. A *managed file* is one that is being managed by a change management system, in this case, *SirLib*.

In managed files, only managed updates should occur. If unmanaged updates are allowed to occur to procedures in managed files, the entire change management scheme can be compromised. This is why it is recommended that you:

1. Convert your managed files to PUBLIC with low (x'0221') privileges.
2. Allocate the files to the *SirLib* subsystem.

Then anyone can read the file, but only the change management system can update procedures in it.

Note: If this is your first time using *SirLib*, you should read "Getting Started" on page 61 before continuing with the detailed portions of the *SirLib* manual.

CHAPTER 3 SirLib Implementation

Managing changes to program code generally means controlling entire procedures or files. *SirLib* manages ***changes***, that is, differences between old and new versions of procedures.

The central construct underlying the *SirLib* system is that of an ***update procedure***. An update procedure is a *Model 204* procedure containing lines of code to be changed in another *Model 204* procedure.

Interspersed with the lines of code are lines containing *SirLib* commands and sequence numbers which tell *SirLib* where in the target procedure the lines to be changed are located. In other systems, update procedures are sometimes referred to as “delta's”, or differences. All update procedures for one file are kept together, separate from the managed file in another *Model 204* procedure file called the “**FixFile**”.

```
 XMPLPROC.F2BALES.PRE_MAIN     XMPLPROC.F2BALES.PRE_MAIN      1 OF 15

====>
=====  * * * TOP OF PROCEDURE * * *
===== ./ * Update generated by ALAN on 12-Ø4-91 at 14:22 from WORKPROC
===== ./ R 03280000 03290000 $ 03280001 00001000
=====                  IF $LEN(%MPROC) GT 30 THEN
=====                     :%SCR.PROC = $SUBSTR(%MPROC,1,29) WITH '*'
=====                  ELSE
=====                     :%SCR.PROC = %MPROC
=====                     MODIFY :%SCR.PROC TO DIM
=====                  END IF
=====                  :%SCR.PROC = $SUBINS(:%SCR.PROC,%LITEM,33)
===== ./ I 03590000           $ 03590100 00000100
=====         FOR %IDX FROM 1 TO %MAX.POS
=====            %LITEM = $LISTINF(%LIST,%IDX)
=====            %MPROC = $SUBSTR(%LITEM,39)
=====            %TEST  = $PROCOPN(%MPROC,%MFILE)
=====            IF NOT %TEST THEN
=====  * * * BOTTOM OF PROCEDURE * * *
```

An Update Procedure

The naming convention for *SirLib* update procedures identifies the target file for the update, the project to which the update links, and the procedure to be updated. Update procedure names must follow the following convention:

```
<filename>.<project>.<proc-name>
```

In the example above, the name of the update procedure being edited identifies the target file as XMPLPROC, and the project as F2BALES. The remainder of the update procedure name identifies the target procedure as PRE_MAIN.

The ./R command in the first line tells *SirLib* to *R*eplace the lines sequenced as 328 through 329 with the following lines of code (up to the next "./" command prefix). The ./I command tells *SirLib* to *I*nsert after the line sequenced as 359 the lines up to the next "./" command. Deletion of lines of code is handled in a similar way with "./D". *SirLib* inserts "./ *" comments at the top of each update procedure specifying the user, date, time and source file for each change.

The other kind of procedure contained in a FixFile is a Control procedure. Control procedures are *Model 204* procedures that contain control data for the change management process. The control data is simply a list of *Projects*.

```
 CONTROL.SIRMON               CONTROL.SIRMON                1 OF 1Ø

====>
=====  * * * TOP OF PROCEDURE * * *
===== _ B4ENV    Environmental probs, CCASYS privs, FSOUTPUT, UTABLES  (AB)
===== _ B4UTABLE Make UTABLE settings valid for Version 2.1.           (AB)
===== _ B4ACCT   Add ACCOUNT to user parms help text.                 (AB)
===== _ 22STATS  New statistics for Version 2.2Ø of the $functions.   (AB)
===== _ B4FIHELP Fix the description of ENQSHR and ENQEXC.            (AB)
===== _ B3CCNMSC Fix miscellaneous bugs, mostly found at CCN.         (AB)
===== _ B4FSON   Point the start procedure to FSON/FSDET.             (AB)
===== _ B4GTBL   Fix the LGTBL reset on exit from subsystems.         (AB)
===== _ F4IMPACT Enhancements suggested at IMPACT '92.                (AK)
===== _ F4SCROLL Make NOSCROLL do a ROLL SCROLL, fix help scrolling   (AB)
=====  * * * BOTTOM OF PROCEDURE * * *
```

Control procedures contain project identifiers

Project identifiers are 8-character names of logical changes which allow physical changes (actual update procedures) to be grouped so they are always applied and backed together. Project identifiers are used as the second qualifier of update procedure names. Project identifiers may also be followed by a line of comment in the CONTROL.xxxxxxxx procedure, as shown above.

Two more concepts central to *SirLib* operations are *Base* and *Sequenced* procedures.

When *SirLib* attempts to apply an update against a procedure it looks first for a procedure of the same name with a prefix of "**BASE.**". *BASE.* Procedures are versions of procedures before any updates have been applied to them through *SirLib*. If a BASE. procedure is found *SirLib* applies the changes as directed to the BASE. version, deleting and replacing the previous executable procedure. If no BASE. version of the procedure is found *SirLib* assumes this is the first managed change to be applied to this procedure, and it creates the BASE. version before changes are applied (by copying the executable version into a BASE. version). *Once a BASE. version of any procedure exists in a file it should never be altered or deleted*. BASE. procedures should only be deleted as part of the Cutover process described later.

Until the first update is applied to a procedure there is no need to create a separate BASE copy of the procedure. The list of BASE procedures in the managed file is the list of procedures that have had updates applied to them. *SirLib* always creates base procedures automatically; no user action is required to create a base.

Sequenced versions of procedures are produced when programmers use *SirLib* facilities to make copies of procedures to work on. Sequenced versions are created automatically and named the same as the target procedure but prefixed with "**SEQ.**". Internal sequence numbers are attached to each line of the original procedure and stored as part of the sequenced copy. When the programmer is finished making changes, this numbered version is used as the comparison procedure for producing an update procedure. Sequenced versions of procedures are kept by programmers for however long changes are being made and update procedures are being generated (update procedures can be regenerated as many times as a programmer wishes). When the programmer is finished with the program changes, and a final version of the update procedure has been generated, the sequenced and working version of the procedure may be deleted.

While this process for generating updates sounds complicated, in practice it is simpler than most other change tracking systems, and far simpler in many ways than not using a change control system at all. The programmer generates working and sequenced copies of a procedure with the Q command, makes any changes needed, then generates the update procedure (the "*differences*") with an X command (XCOMPARE). The update procedure is applied as part of a project when all parts of that project are completed.

In *SirLib*, the most important procedures are the BASE.<procname> and the update procedures that apply to that BASE. procedure. If actual executable procedures are deleted they can be recreated by applying the updates to BASE. versions of the procedures. This is how updates are applied when a file is being reconfigured.

Change tracking and reporting can be done using the update procedure names alone—they supply the exact number of changed programs in a release, and coupled with the number of lines within the update procedures, managers can tell the number of lines of new or replacement code generated, and the date and time each update was completed.

Programmers need never be given update access to the actual production copies of procedure files. Development and testing is done by whatever method a shop uses. The assumed standard is for shops to define development APSYs with grouped procedure files, though this method is not a prerequisite for *SirLib* functioning.

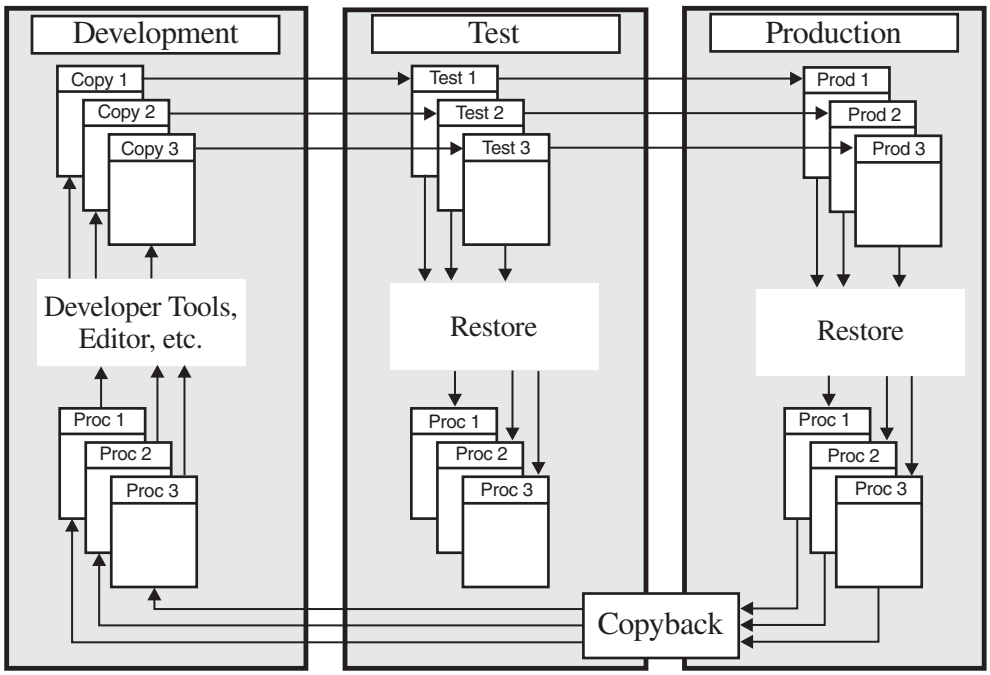
When a programmer finishes working on a procedure an update is generated instead of the entire procedure being copied to a staging or production file. Once all update procedures are generated for a release, a manager reviews the updates to make sure they conform to local coding standards and that they are linked to projects existing in the control procedure. This level of monitoring, which used to be difficult if not impossible to do, is a simple task under *SirLib*, because the update procedures tend to be small, and

because the internal comments clearly indicate who generated the new code, where it came from and when it was created. The manager applies the updates via option 2 in *SirLib*, and the production version of the procedure is ready for testing or distribution.

Another option is to distribute the FixFile itself. A shop with multiple *Model 204* online regions, can run a copy of *SirLib* in each region. Update procedures can be distributed in the FixFile and applied on a region-by-region basis; this allows for faster backouts in production regions, and removes the complications of distributing each managed file.

3.1 SirLib Operations Overview

Without *SirLib*, User Language change control might look like this:

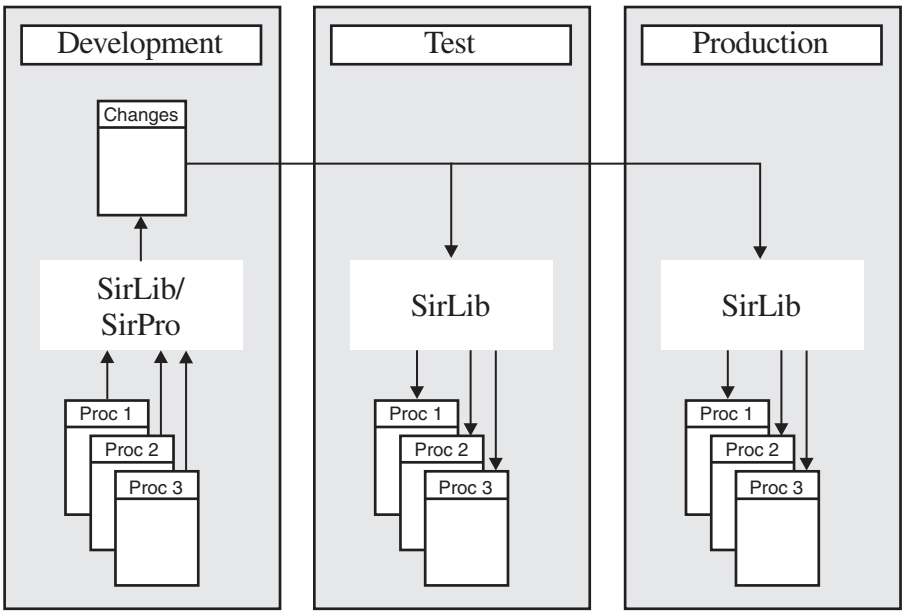


Typical Model 204 procedure promotion.

Typically, programmers make backup copies of procedures they are about to work on, either in the development file or in a “backup” file set aside for this purpose. They then work on the original procedure in the development file, integrating their changes either in the development file or when updates are sent to a test environment. Changes are promoted from development to test and from test to production either on a procedure-level or a file-level basis.

As shown in the illustration, often each file requires its own distribution or “staging” copy in each environment, and a mechanism is usually provided for copying procedures back from production to test and from test to development (the backward migration paths at the bottom of the picture).

On large projects the flow of change promotion can become extremely complex, with multiple development files and online environments, and various levels of integration both within and across onlines. Obviously the more complex the development path, the greater the difficulty of getting into production an error-free User Language system.



Change Management using SirLib

The figure above shows a change promotion scheme using the *SirLib* system. In the simple flow diagram shown above changes for the various procedure files are all generated into a single FixFile (here called Changes) in the development environment. The Changes file is distributed or shared across onlines, and the updates are applied in each region by a local copy of *SirLib*.

The majority of configuration management tasks are invisible to programmers using *SirLib*. Programmers work in *SirPro*, which acts as a front-end to the *Model 204* editor and to the *Model 204* command structure. *SirPro* gives the User Language developer an SPF-like “toolbox” with which to work, supplying prefix commands for copying, deleting, moving, renaming, editing and browsing User Language procedures in a selected file. In addition, when *SirLib* is managing changes, special prefix commands are supplied within *SirPro*, that tell the *SirLib* system that a *managed update* is taking place. The commands are:

Q SEQUENCE. The Q command tells *SirLib* to generate working and “sequenced” copy of a procedure.

X XCOMPARE. Generates an update procedure by comparing the working and sequenced procedure copies. Output is a procedure containing the differences.

N NEW. Generates an update procedure for a new procedure.

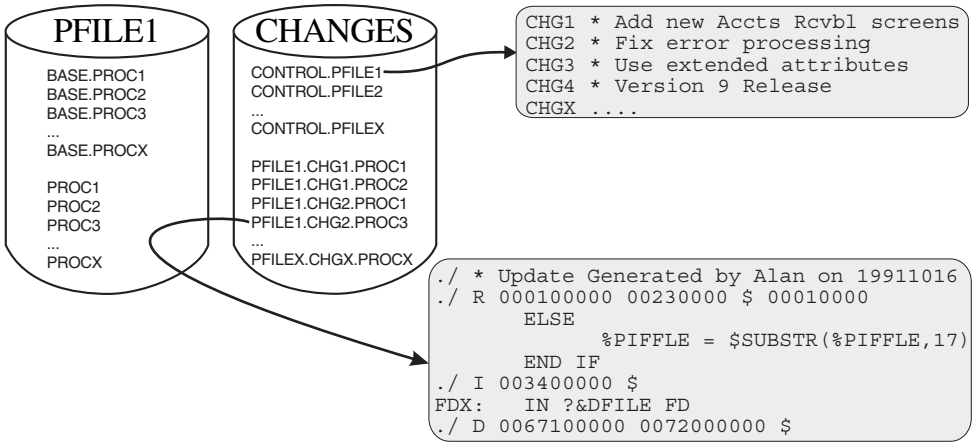
K KLOBBER. Generates an update procedure for a logical delete of a procedure.

- **Z** RESEQUENCE. Renumbers the internal sequence numbers for a procedure that has had many changes applied to it without the file being cutover.

These commands are explained in more detail in the next section.

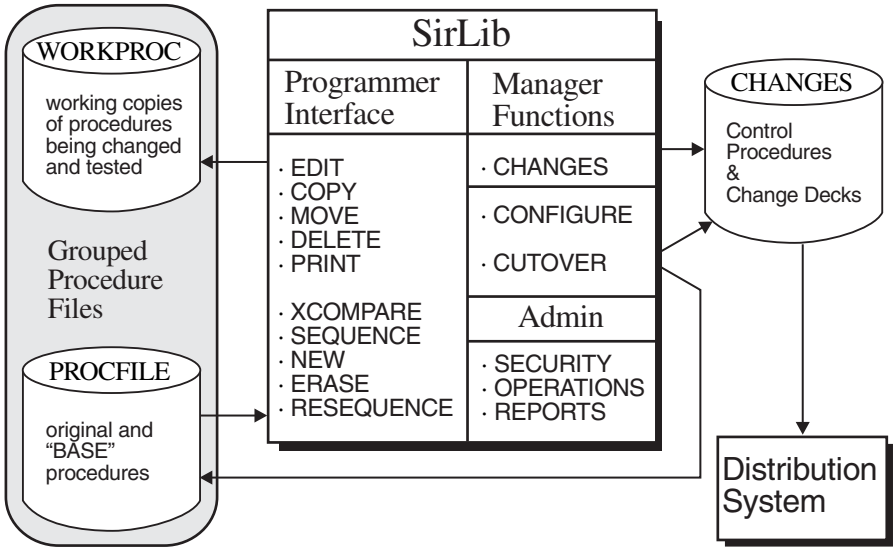
Note in the previous picture, all the promotions paths are simplified. This is because *SirLib* allows updates to all managed files to be stored, managed and promoted within the same FixFile. Note also the elimination of the “copyback” path from the previous illustration. This path is generally a violation of a managed environment, and is tolerated because of the need to generate emergency fixes against production code, and because of the occasional requirement to re-align development and production procedures. *SirLib* eliminates this path by providing the tools to let users know the exact state of production code, and to duplicate the “update level” of production in any other environment.

Programming a managed update consists of making a working and sequenced copy of a procedure using the “**Q**” command, making program changes to the working copy then executing an “**X**” command against the copy to produce an update procedure. The “**X**” command compares the working copy which has changed, to the sequenced copy which will not have changed. The resulting update procedure is the differences between the two.



Update procedures linked to project via naming convention

Once an update procedure is produced the programmer can delete the working copy of the procedure and the matching sequenced version; the *SirLib* system generates the new production procedure by applying the update(s) against the BASE procedure. This is done by a programmer, project leader or administrator via the CONFIGURE option in *SirLib*.



SirLib Functions

Management activity in the *SirLib* system begins when a functional change is identified in a particular file in the local application. In the *SirLib* Project screen (option 1), a new line is entered containing an 8-character project name and an optional description. PF key access allows an unlimited amount of optional documentation to be attached to a Project. The physical order of these project identifiers should not be altered once updates are linked to the identifiers, as this sequence specifies the order in which updates are applied. Updates that effect the same line of code will not apply correctly unless previous updates have been applied.

```
------------------------------ * * * SIRLIB * * * ------------------------------

===>
 Change                    Description                        (FILE: SIRMON)
--------------------------------------------------------------------------------
_ B4ENV    Environmental probs, CCASYS privs, FSOUTPUT, UTABLES  (AB)
_ B4UTABLE Make UTABLE settings valid for Version 2.1.           (AB)
_ B4ACCT   Add ACCOUNT to user parms help text.                  (AB)
_ 22STATS  New statistics for Version 2.20 of the $functions.    (AB)
_ B4FIHELP Fix the description of ENQSHR and ENQEXC.             (AB)
_ B3CCNMSC Fix miscellaneous bugs, mostly found at CCN.          (AB)
_ B4FSON   Point the start procedure to FSON/FSDET.              (AB)
_ B4GTBL   Fix the LGTBL reset on exit from subsystems.          (AB)
_ F4IMPACT Enhancements suggested at IMPACT '92.                 (AK)
_ F4SCROLL Make NOSCROLL do a ROLL SCROLL, fix help scrolling    (AB)




-------------------- ULSPF.Ø18: Enter new information and confirm with PF12.
1/Help     2/Document  3/Quit     4/Insert
7/Up       8/Down      9/Repeat                                  12/Commit
```

Project screen

The Project option allows entry of Project names, and allows PF key access to a documentation feature for the project.

Entry fields on the Projects screen are:

Status If this column is blank (or shows the default pad character, the underscore, the project is active and changes associated with this project will be included in subsequent file reconfigures and cutovers. To comment out a project, enter an asterisk ("*") in this field. Care should be taken in commenting out projects that have dependencies in projects that are implemented after them.

Project ID The Project ID is an alphanumeric identifier of 1 to 8 characters that links together change decks containing related changes. The project ID should conform to some meaninful pattern as established at the site. For instance, all bug fixes could be prefixed with "B" and new features prefixed with "F", followed by a version number and some mneumonic.

Description Up to 72 characters of text can be added to the project. This descriptive text is displayed on the reconfigure screen and cutover screen, and in the SIRLIB reports. Extra text can be added to a project via PF2 and this text is stored as data records in SIRLIBD.

In a highly controlled environment the project leader may require that no changes occur in a file unless a project is entered first via the Project facility (options are provided to protect the ability to define Projects). In a less structured project, updates may be produced as programmers need to build them, and a project leader may later decide which will be included in a particular release by adding a project that links the updates in.

Along with making distribution files smaller this allows changes to be distributed and applied by *SirLib* directly in a production region. Generally, all changes for all procedure files can be kept in a single file, greatly reducing complex distribution problems. Operating in this manner allows changes to be backed out much faster. It also allows releases to be shipped to all sites and applied on a region-by-region or project-by-project basis. Status of the current configuration can be viewed in *SirLib* screens and also by reviewing the procedure comments placed by *SirLib* inside each proc that has had an update applied to it. Change status can also be seen in *SirLib* reports.

CHAPTER 4 SirLib — A Programmer's Reference

```
------ FILE: SIRPRO ------ ULSPF303/2.2.0/CMS ----- 93-03-31 11:18:04 ------
==>                                                  Total Procs = 25
Sel     Procedure Name                  Account      Bytes   Date     Time
  1    PUPR-CCAGRP                     ALAN          26823 93/03/30  8:44:59
  2    PUPR-CCASTAT.FILE               ALAN          25293 93/03/30  8:45:00
  3    PUPR-CCASTAT.USER               ALAN          19678 93/03/30  8:45:02
  4    PUPR-COMMAND                    ALAN          17644 93/03/30  8:45:04
  5    PUPR-COMP.SHIJI                 ALAN          18929 93/03/30  8:45:05
  6    PUPR-COPY                       ALAN          10288 93/03/30  8:45:06
  7    PUPR-COPY.CHECK                 ALAN           3986 93/03/30  8:45:06
  8    PUPR-COPY.SHIJI                 ALAN          37391 92/08/14  9:09:25
  9    PUPR-COPY2                      ALAN          18671 93/03/30  8:45:07
  10   PUPR-DELETE.CHECK               ALAN           5319 93/03/30  8:45:08
  11   PUPR-DELETE.SHIJI               ALAN          34747 93/03/30  8:45:09
  12   PUPR-DEL1                       ALAN          19552 92/08/14  9:10:02
  13   PUPR-DEL2                       ALAN          21340 93/03/30  8:45:10
  14   PUPR-EDIT1                      ALAN          20437 93/03/30  8:45:11
  15   PUPR-EDIT2                      ALAN          30427 93/03/30  8:45:12
  16   PUPR-ERASE.SHIJI                ALAN          12098 92/08/14  9:09:02
  17   PUPR-KILL.SHIJI                 ALAN          16985 93/03/30  8:45:14
------------------------------------------- ULSPF.120: Updates not allowed.
1/Help   2/Sort-Name    3/Quit     4/Sort-User   5/Sort-Date   6/Sort-Size
7/Up     8/Down         9/Repeat  10/Refresh    11/*UPPER     12/FULLNAME
```

SirPro Procedure Editor Screen (EDIT)

The *SirPro* screen shown above is the main working environment for the programmer in the *SirLib*-controlled environment. This screen, and others like it, provide access to all procedure activity in *Model 204*. Details on the use of *SirPro*, including managed update commands, are contained in the &SPROUG.. Managed update prefix commands are the only part of *SirPro* discussed in this document.

The X, Q, N, K and Z prefix commands are used to generate 'working', base and sequenced copies of procedures and to generate updates from those copies. The programmer types one of these managed update commands to the left of the selected procedure and presses ENTER. *SirPro* then presents one of the managed update screens to get further details on the managed update. Those screens are shown in the following detailed sections.

4.1 Q (SEQUENCE) Command

“Sequencing” is the managed update method by which programmers make copies of procedures to work on. In other systems, programmers make backup copies of procedures and then work on the original. In *SirPro* programmers never alter the original program code, but execute “Q” commands to get working and sequenced copies from which update procedures can be generated.

Updates must be done in a file other than the original, which is why procedure groups are recommended for development apsys.

```
--- Sequence Procedure ----- ULSPF303/2.2.0/CMS ----- 93-03-31 15:00:16 ----
==>

  Input procedure
   Proc file             ==> SIRPRO
   Procedure             ==> PUNP-SMARTBROWSE

  Output procedures
   Proc file             ==> DEVPRO              Password ==>
   Unsequenced procedure ==> PUNP-SMARTBROWSE
   Sequenced procedure   ==> SEQ.PUNP-SMARTBROWSE


  The FixFile must contain control procedure CONTROL.SIRPRO
  and all updates for file SIRPRO

   FixFile               ==> SIRFIXES            Password ==>


  Replace existing procedures? (Y/N) ==> N
--------------------------------------------------------------------------------
1/Help              3/Quit
```

SirPro screen for generating SEQ and working procedure copies

The Q command tells *SirLib* to generate both a copy of the selected procedure (the copy that the programmer will work on) and a “sequenced” copy of the procedure (a copy containing sequence or line numbers), which will later be used as comparison input for the Xcompare command. Building these copies is always the first programming step in generating a managed update. The Q prefix command presents a screen (*Sequence Procedure*) with the following input fields:

Input procedure

The following two fields identify the source procedure, i.e. the version of the procedure before it is changed. This information is copied from the entry on the SirPro procedure editor screen.

Proc file — Name of the *Model 204* file containing the source procedure.

Procedure — Name of the source procedure. This is the procedure that was subject to a “Q” prefix command on the *SirPro* EDIT screen.

Output procedures

SirLib is designed to work in a *Model 204* subsystem (APSY) context using procedure groups. The "Q" command creates two output procedures that must be in a different file from the source procedure. One of these procedures is used as a working copy, while the other procedure is used to determine any changes made to the working copy. Both of these procedures are formed by applying any staged updates to the indentified source procedure.

Proc file — Name of the *Model 204* file to contain the two procedures produced by the *SirLib* "Q" command.

Password — If a password is required for update access to the file it must be entered here.

Unsequenced procedure

Name of the procedure that will become the working copy for developing and testing changes. This defaults to the name of the source procedure.

Sequenced procedure

Name of the procedure that will contain a sequenced version of the working copy, **before** any changes have been made. The sequenced procedure may be used by the *SirLib* "X" command to prepare an update procedure reflecting any changes made to the working copy. The default procedure name for sequenced procedures is the unsequenced procedure name prefixed with **SEQ.**:

```
SEQ.<unsequenced procedure name>
```

This default prefix may be overridden if it conflicts with local conventions.

Location of control and update procedures

SirLib uses a naming convention to identify any updates that should be applied to the source procedure before generating a working copy. FixFile is searched for a "Control Procedure" whose name must be the name of the input procedure file prefixed with **CONTROL.**:

```
CONTROL.<input procedure proc file name>
```

The Control Procedure identifies all projects with updates to procedures in the input procedure file. For each such project, FixFile is checked for Update Procedures that should be applied to the source procedure.

FixFile — Name of *Model 204* file containing control and update procedures used to build the output procedures. Default is **SIRLIBP**.

Password Read access is required for the control procedure any update procedures. If a password is required for read access it must be entered here.

Replace existing procedures?
Overlaying a working copy or sequenced copy of a procedure can cause updates in progress to be lost. The default for this field is **N**, which directs the Q command to not overwrite an existing procedure. If a user wishes to overwrite existing procedures this indicator should be set to **Y**.

When the "Q" command is executed, *SirLib* looks in the source file for a procedure named **BASE.<procname>**, where **<procname>** is the name of the selected procedure. If no such procedure is found then *SirLib* copies the procedure into the target procedure in the Output file. In addition, *SirLib* generates a sequenced copy of the procedure, named whatever the programmer specified in the *Sequenced Copy* field on this screen. The sequenced copy of the procedure is identical to the unsequenced (or working) copy, except that each line is prefixed with a sequence number.

If *SirLib* does find a **BASE.<procname>** procedure then it looks for the **CONTROL.<filename>** procedure in the specified FixFile. A control procedure of the specified name must exist or the Q command fails. Noting the active projects in the control procedure, *SirLib* finds all update procedure names that match the pattern:

```
<file>.<project>.<procname>
```

where *<file>* is the name of the input file for this operation, *<procname>* is the name of the selected input procedure, and *<project>* is one of the identifiers in columns 1 to 8 of the control procedure. All update procedure names that conform to this pattern are sorted by their project and then by the sequence line numbers they effect. This aggregate update is then applied to **BASE.<procname>** to produce the sequenced and unsequenced output procedures. In other words, the programmer doesn't just get a copy of the procedure they request, they get a generated procedure composed of the BASE procedure with all updates applied to it.

If a project name is commented out in the control procedure it is not included in the resulting sequenced and unsequenced output procedures. A Project name is commented out by placing an asterisk ("*") in front of it in the control procedure. (This is how working versions of production procedures can be generated in development regions if the development region has active projects that do not exist yet in production).

The sequenced version of a procedure should never be changed, as it is the "before" image for the XCOMPARE that will eventually generate the update procedure. Sequence numbers should not concern programmers for the most part, though there are a few times when it is worth knowing something about how they work.

The "Q" command automatically begins sequencing at 10000 and increments each line in the BASE.<procname> procedure by 10000. As changes are generated, the XCOMPARE function generates new line numbers for lines of code being inserted and

replaced. The XCOMPARE algorithm attempts to number the first new line of code beginning with a sequence number 1 greater than its starting point in the existing sequence (this applies to Inserts and Replaces: it is irrelevant for Deletes). Subsequent new lines which are part of the same update are incremented by a power of 10 less than the last sequence of numbers for the section of code. Later changes that apply to the same section again begin numbering at their starting sequence number plus 1, and continue to increment at the next lowest available power of 10.

Fortunately, the programmer doesn't have to understand any of this in order for it to work. The lower-order digit(s) of the sequence numbers in a SEQ procedure however will indicate the number of times a section of code has been changed. The sequence numbers themselves will indicate whether the XCOMPARE will be able to “fit” changes into the same section of code again. For example, a section of code with sequence numbers that look like this:

```
003451001
003452001
003453001
```

indicates that changes will still “fit” in the hundreds and tens columns. If an update procedure attempts to insert 3 lines of code after line 003452001, the resulting code will be numbered like this:

```
003451001
003452001
003452101
003452201
003452301
003453001
003454001
```

Replace (“./R”) commands operate slightly differently than Insert, but the outcome looks very similar. When changes no longer “fit” within the sequence numbers XCOMPARE will still generate an update procedure for this section of code, but the update procedure will be unnecessarily long and *SirLib* might lose its ability to detect update collisions. In this case, a Resequence is required, as described later in this chapter.

Once an initial update has been coded and the update procedure saved in the FixFile, all subsequent “Q” commands against the same procedure will generate copies containing the update, and sequence numbers in output sequenced versions of the procedure will reflect the insertion of the update code.

4.2 X (XCOMPARE) Command

Prefixing a procedure by an “X” in *SirPro* tells the *SirLib* system to compare the specified procedure against a matching “sequenced” procedure, generating another procedure that containing the differences between the two. The X prefix command presents a *XCOMPARE Screen*:

```
--- Build Update Deck ------ ULSPF303/2.2.0/CMS ----- 93-03-31 15:02:42 ----
==>

  Unsequenced input procedure
   Proc file          ==> DEVPRO
   Procedure          ==> PUNP-SMARTBROWSE

  Sequenced input procedure
   Proc file          ==> DEVPRO                 Password ==>
   Procedure          ==> SEQ.PUNP-SMARTBROWSE

  Output update procedure
   FixFile            ==> SIRFIXES               Password ==>
   Project Name       ==> F4BATCH
   Target File        ==> SIRLIB

  Replace existing update? (Y/N)  ==> N
  Enter editor for update? (Y/N)  ==> N

  Synchronization count           ==> 2
  Unlock this procedure? (Y/N)    ==> Y
-------------------------------------------------------------------------------
1/Help              3/Quit
```

SirPro XCOMPARE screen

Unsequenced input procedure

Identifies the procedure to be examined for changes, which was the object of an “X” command from a *SirPro* procedure list screen.

Proc file — Protected field showing name of input procedure.

Procedure — Protected field showing name of unsequenced input procedure.

Sequenced input procedure

Identifies the sequenced version of the input procedure, showing its contents prior to any changes.

Proc file — Name of *Model 204* file containing the sequenced input procedure. Deafults to the same file as the input unsequenced procedure.

Password — Read access is required for the sequenced input procedure. If the file is not already open with sufficient privileges and a password will be required, it should be entered here.

Procedure — Name of the procedure containing a sequenced representation of the unsequenced input procedure before any changes were applied. Defaults to the name of the unsequenced input procedure with a prefix of “SEQ.”.

Output update procedure

Identifies the location and name of an update procedure that will contain the changes that have been made to the sequenced version of the source procedure.

FixFile Name of *Model 204* file to contain the update procedure produced by this invocation of XCOMPARE.

Password Write access is required for the output update procedure. If the file is not already open with sufficient privileges and a password will be required, it should be entered here.

Project name

This is the 8-character project name to which this update should be linked.

The file administrator may have required that updates for this file only be linked to existing projects, in which case the entered project name is checked to make sure it exists in the control procedure. If this setting is not on, updates may be created, and the project entered in the control procedure later.

Target File This is the production file containing the base procedure against which this update will apply.

Replace existing update

Entering “Y” in this field allows the user to overwrite an existing version of the update procedure with the latest changes. Programmers are always permitted to overwrite update procedures created under their userid when this switch is set to “Y”. The file administrator may have specified in the Administration options that programmers may not overwrite each other's update procedures either in this file or within the entire system. If this option is set, and an update procedure exists of the same name as the one being created in the target file, and the userid who created the update procedure (or last updated it) is not the same as the current user's id, then the generation of the update will not be permitted.

Enter Editor for update

Entering “Y” in this prompt places the generating user into an edit session on the new update procedure. The update procedure will have already been stored before the edit session is invoked, so PF3 or QUIT may be used to exit the edit session without losing the update procedure.

Synchronization count

This field allows programmers to set the number of lines that will be compared before the sequenced and unsequenced versions of the procedures are considered to be back in sync, and further lines are no longer to be generated to the output update procedure (at least until the next mis-match). The default sync count value is 2, and this will work well for the vast majority of cases.

Increasing the sync count may reduce the number of difference lines found, resulting in smaller update procedures, or it may have no effect at all beyond a minor performance penalty in the XCOMPARE operation. In a very few cases it may produce larger output procedures. An entered value of "*" tells XCOMPARE to repeat the compare operation, increasing the sync count from 1 until it hits a value that no longer produces a smaller output procedure. This is the worst-performing option for running XCOMPARE, but will almost always generate the smallest possible output procedure. Unless changes are massive and many, the size of the output procedure is fairly irrelevant, and playing with the sync count has little utility.

Unlock this procedure?

Entering "Y" in this field makes the procedure available to other programmers in systems or files where procedures are locked to one updating user at a time. Any value but "Y" allows the programmer to generate an update procedure but will retain the exclusive lock on the procedure (such as when a test integration is being run, and the programmer doesn't wish to relinquish control of the procedures being updated). If procedure locking is not turned on, the lock/unlock information is still maintained for historical reporting, but this switch has no actual effect in locking procedures. See Administration options for Procedure Locking options.

4.3 N (NEW) Command

Prefixing a procedure by an N in the *SirPro* edit screen tells the system to generate an update procedure, similar to that produced by X, but containing *all* lines of the specified procedure, because it is a new procedure to the system. The N prefix command presents a *Create New Procedure* screen:

```
--- Create New Proc -------- ULSPF303/2.2.0/CMS ----- 93-03-31 15:03:50 ----
==>

  New procedure name
     Proc file        ==> DEVPRO
     Procedure        ==> PUNP-SMARTBROWSE



  Output update procedure
     FixFile          ==> SIRFIXES              Password ==>
     Project name     ==> F4BATCH
     Target file      ==> SIRLIB

  Replace existing update?  (Y/N) ==> N
  Enter editor for update?  (Y/N) ==> N




--------------------------------------------------------------------------------
1/HELP              3/QUIT
```

Generating a New update procedure

New procedure name

Identifies the new procedure, which was the object of an “N” command from a *SirPro* procedure list screen.

Proc file Name of *Model 204* file containing the new procedure.

Procedure Name of the new procedure.

Output update procedure

Identifies the location and name of an update procedure that will represent the creation of a new procedure.

FixFile Name of *Model 204* file to contain the update procedure.

Password Write access is required for the output update procedure. If the file is not already open with sufficient privileges and a password will be required, it should be entered here.

Project name

This is the 8-character project name to which this update should be linked. The file administrator may have required that updates for this file only be linked to projects previously defined in the control

procedure. In this case the project entered is verified to see that it exists before the update is generated. Otherwise the project name is used to determine the name for the update procedure.

Target File This is the production file containing the base procedure against which this update will apply.

Replace existing update

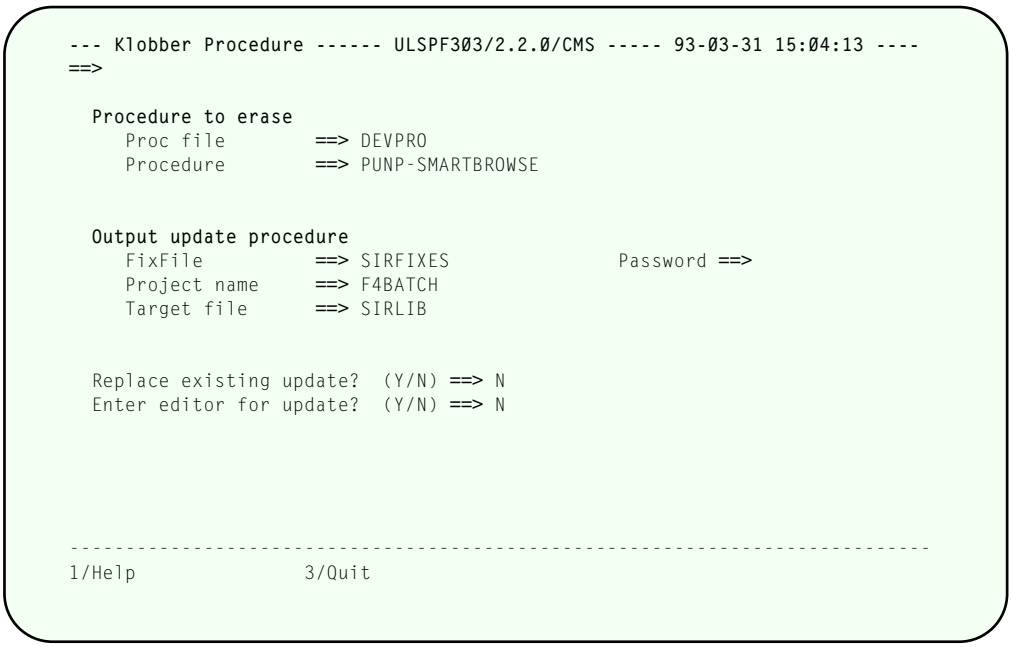
Entering “Y” in this field allows the user to overwrite an existing version of the update procedure with the latest changes. Programmers are always permitted to overwrite update procedures created under their userid when this switch is set to “Y”. The file administrator may have specified in the Administration options that programmers may not overwrite each other's update procedures either in this file or within the entire system. If this option is set, and an update procedure exists of the same name as the one being created in the target file, and the userid who created the update procedure (or last updated it) is not the same as the current user's id, then the generation of the update will not be permitted.

Enter Editor for update

Entering “Y” in this prompt places the generating user into an edit session on the new update procedure. The update procedure will have already been stored before the edit session is invoked, so PF3 or QUIT may be used to exit the edit session without losing the update procedure.

4.4 K (ERASE) Command

Prefixing a procedure with a “K” command in the *SirPro* edit screen directs *SirLib* to generate an update procedure that will cause the deletion of the indicated procedure. Managed update deletes are always logical deletes, allowing deleted procedures to be recovered at any time when the file is reconfigured. For this reason, an empty (zero-line) version of the erased procedure is left in the file, and should not be physically deleted. The K prefix command presents an *Erase Procedure* screen:



Generating an Erase (a logical delete) of a procedure in SirPro

Procedure to erase

Identifies the procedure to be erased, which was the object of a “K” prefix command from a *SirPro* procedure list screen.

Proc file Name of *Model 204* file containing the procedure.

Procedure Name of procedure to be deleted.

Output update procedure

Identifies the location and name of an update procedure that will represent deletion of the procedure.

FixFile Name of *Model 204* file to contain the update procedure.

Password Write access is required for the output update procedure. If the file is not already open with sufficient privileges and a password will be required, it should be entered here.

Project name

This is the 8-character project name to which this update should be linked. The file administrator may have required that updates for this file only be linked to projects previously defined in the control procedure. In this case the project entered is verified to see that it exists before the update is generated. Otherwise the project name is used to determine the name for the update procedure.

Target File This is the production file containing the base procedure against which this update will apply.

Replace existing update

Entering “Y” in this field allows the user to overwrite an existing version of the update procedure with the latest changes. Programmers are always permitted to overwrite update procedures created under their userid when this switch is set to “Y”. The file administrator may have specified in the Administration options that programmers may not overwrite each other's update procedures either in this file or within the entire system. If this option is set, and an update procedure exists of the same name as the one being created in the target file, and the userid who created the update procedure (or last updated it) is not the same as the current user's id, then the generation of the update will not be permitted.

Enter Editor for update

Entering “Y” in this prompt places the generating user into an edit session on the new update procedure. The update procedure will have already been stored before the edit session is invoked, so PF3 or QUIT may be used to exit the edit session without losing the update procedure.

4.5 Z (RESEQUENCE) Command

The sequence numbers generated by *SirLib* are for the most part of no concern to User Language programmers. However, if a large number of update procedures are generated which effect the same areas of code in a procedure, further changes to the same areas may result in unnecessarily large update procedures. In addition, the process of untangling overlapping updates with *SirLib* will become more complex. This is should be a rare event, however if it does occur, a manager need only generate a resequence procedure by using the "Z" prefix command. The Z command presents a *Resequence Procedure* screen:

```
--- ReSequence Proc -------- ULSPF303/2.2.0/CMS ----- 93-03-31 15:04:34 ----
==>

  Resequence procedure
     Proc file        ==> DEVPRO
     Procedure        ==> PUNP-SMARTBROWSE


  Output update procedure
     FixFile          ==> SIRFIXES              Password ==>
     Project name     ==> F4BATCH
     Target file      ==> SIRLIB


  New starting number             ==> 10000
  New increment                   ==> 10000
  Replace existing update? (Y/N)  ==>     N
  Enter editor for update? (Y/N)  ==>     N



------------------------------------------------------------------------------
1/HELP              3/QUIT
```

Resequencing a procedure in SirPro

Resequence procedure

Identifies the procedure to be resequenced, which was the object of a "Z" prefix command from a *SirPro* procedure list screen.

Proc file Name of *Model 204* file containing the procedure.

Procedure Name of procedure.

Output update procedure

Identifies the location and name of an update procedure that will cause resequencing of the procedure.

FixFile Name of *Model 204* file to contain the update procedure.

Password Write access is required for the output update procedure. If the file is not already open with sufficient privileges and a password will be required, it should be entered here.

Project name

This is the 8-character project name to which this update should be linked. The file administrator may have required that updates for this file only be linked to projects previously defined in the control procedure. In this case the project entered is verified to see that it exists before the update is generated. Otherwise the project name is used to determine the name for the update procedure.

Target File This is the production file containing the base procedure against which this update will apply.

New starting number

The default is to begin renumbering with 10000.

New increment

The default is to renumber in increments of 10000.

Replace existing update

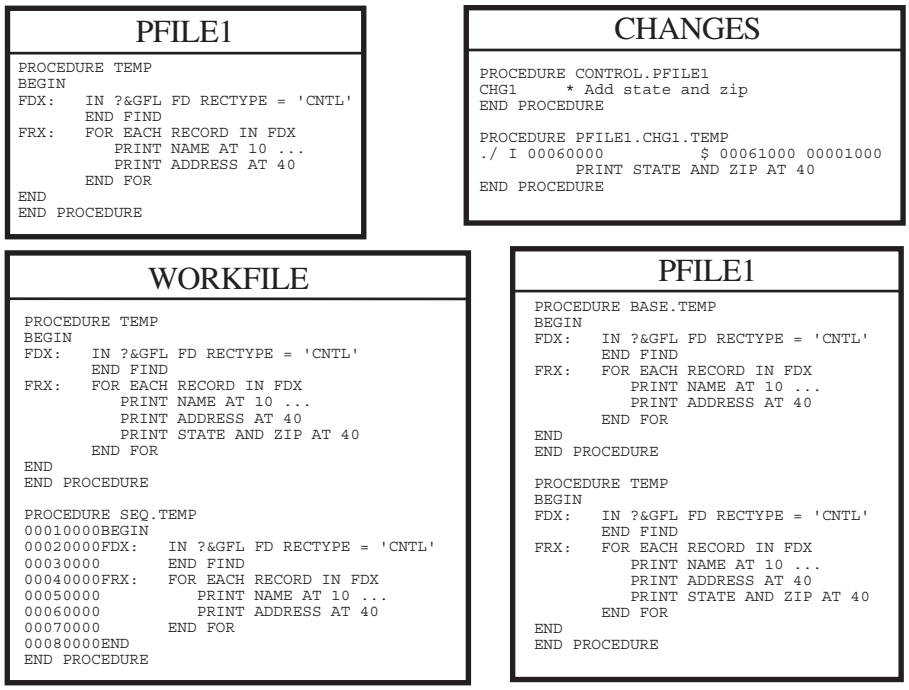
Entering “Y” in this field allows the user to overwrite an existing version of the update procedure with the latest changes. Programmers are always permitted to overwrite update procedures created under their userid when this switch is set to “Y”. The file administrator may have specified in the Administration options that programmers may not overwrite each other's update procedures either in this file or within the entire system. If this option is set, and an update procedure exists of the same name as the one being created in the target file, and the userid who created the update procedure (or last updated it) is not the same as the current user's id, then the generation of the update will not be permitted.

Enter Editor for update

Entering “Y” in this prompt places the generating user into an edit session on the new update procedure. The update procedure will have already been stored before the edit session is invoked, so PF3 or QUIT may be used to exit the edit session without losing the update procedure.

CHAPTER 5 SirLib — Update Example

Notice that sequenced versions of procedures do not apply to New, Erase or Resequence operations. Also note that Q is the managed update equivalent of a Copy, just as N is the equivalent of Create and K is the equivalent of Delete. XCOMPARE has no direct equivalent in the unmanaged command structure, though coupled with later reconfiguration using the output update procedure it is analogous to moving finished changes to a staging area. Resequencing is of course a special purpose function with no unmanaged corollary.

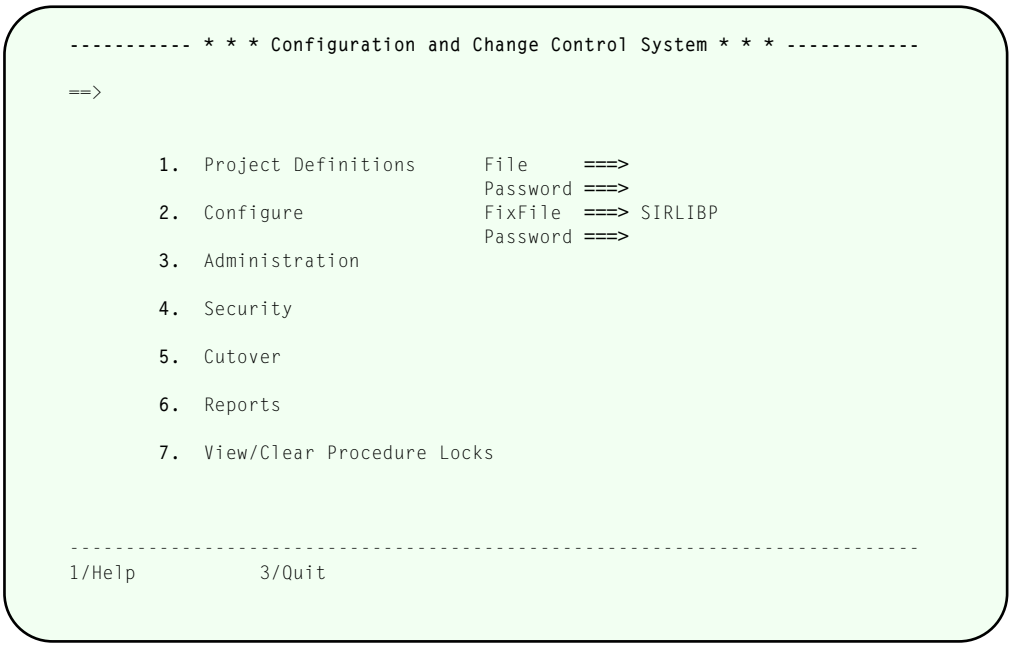


How a change progresses through SirLib

In the example above, procedure TEMP exists in file PFILE1 (upper left). The programmer uses the “Q” command to copy TEMP into WORKFILE, getting both a working and a “SEQ” version of the procedure. A single line of new code is inserted in TEMP (the line “PRINT STATE AND ZIP AT 40”). Using XCOMPARE, the update is generated to file CHANGES, and named PFILE1.CHG1.TEMP. A project leader defines the project name CHG1 in CONTROL.PFILE1. Finally, *SirLib*'s Reconfiguration function, described in the next section, is used to create a “BASE” version of the procedure, so the update can be backed out if necessary, and the original procedure TEMP is deleted. The update procedure PFILE1.CHG1.TEMP is applied to BASE.TEMP to create the new executable procedure TEMP. The final state of PFILE1 is shown lower right.

CHAPTER 6 SirLib Change Control

SirLib functions are accessed through a main menu in the *SirLib* APSY (Selection 5 from the UL/SPF screen). The main menu also allows specification of a File and FixFile. The File field specifies the procedure file targeted for the requested activity. The FixFile field specifies the *Model 204* procedure file that contains the **CONTROL.<filename>** procedure and all update procedures for the selected File.



SirLib Main Menu

Options 1, 2 and 5 require a File to be input. If FixFile is *not* specified for Option 1, 2 or 5, the default FixFile as defined in the Administration option will be used. If FixFile *is* specified for these options, the specified FixFile overrides the default. This allows, for instance, files to be reconfigured from different sets of update procedures in different FixFiles. *SirLib*'s default FixFile is always SIRLIBP.

Change Management functions are:

1. **Project Definitions**: Add, delete or change the Project identifiers in the FixFile assigned to a managed file.

2. **Configure**: Apply and backout changes to files (that is; change the configuration of the file).

3. **Administration**: Administration functions define system defaults and file-specific overrides for those defaults, for security, procedure stamping, Administration ID assignment and other “profile” options. This function is also used to remove files from the *SirLib* system.

4. **Security**: Defines access to *SirLib* main menu options and to the Resequence command (“Z”) in *SirPro*. Security settings only apply if SECURE is set to “Y” in the Administration function for the file (or for the system, if the file is not specifically set).

5. **Cutover**: Does a large-scale clean-up of BASE. Procedures, update procedures, Project identifiers and internal procedure stamps. This option is used infrequently; perhaps between major releases of an application.

6. **Reports**: Access to *SirLib* reports.

7. **View/Clear Procedure Locks**: Access to a display screen of all procedures (optionally restricted by file) that are currently checked out. The “checked out” procedures are displayed even if the administration option for multiple updates is turned on, and therefore no procedures are really locked.

6.1 Project Definition

SirLib is a file-based system. It is not specifically aware of the subsystem structure of a site's applications. This allows *SirLib* to manage applications that have a single procfile-to-APSY relationship for all their APSYs, as well as those that run many APSYs from a single file or have multiple procedure files per ASPY.

SirLib uses two pieces of information to track updates and configuration status for a file. One way is via a single FILE record, kept in SIRLIBD, which holds security and administrative information for the file. The other is the file's Control procedure which is stored with the file's update procedures in a FixFile. Control procedures are named ***CONTROL.<filename>***, where *<filename>* is the name of the managed file. Control procedures contain a single line for each Project name. A Project is a logical change. Update procedures are linked to projects by naming convention. Update procedures are named:

```
<filename>.<project>.<procname>
```

The first qualifier tells *SirLib* what file the update applies to. The second qualifier indicates the project to which the update is linked. The third qualifier is the procedure to be changed.

The project name qualifier provides a mechanism for grouping changes, and controlling and linking their application and backout. When a project is added to a Control procedure, the whole set of changes linked to that project becomes included in any reconfiguration of that file. If the project is commented out or deleted, all update procedures whose names contain that project name are “backed out”; that is, not applied to the BASE procedure(s) during the next reconfiguration.

Control procedures may be edited directly in the *Model 204* editor by users who have been given access to the FixFile (FixFiles should be private files). However, the format of the lines of the procedure must be carefully maintained. Each line of the CONTROL procedure must have a Project identifier in column 1-8 (left-justified and blanks to the right if shorter than 8 characters). Column 9 must be blank and columns 10 through 72 may contain a comment. The Project Identifier must be in upper case. The optional comment may be in mixed case. If a Project name begins with an asterisk (“*”), *SirLib* considers the entire line a comment. Updates are backed out by commenting-out project names in this way, and reconfiguring a file. *SirLib* skips the commented line, applying all previous and subsequent updates.

The proper way to maintain Projects is via Option 1 in *SirLib*. This screen allows users to add, change and delete Project identifiers. It verifies the format of the Project identifier, automatically converting it to upper case. To access this screen, a file must be specified on the main menu: Project identifiers are always file-specific, as are the Control procedures in which they are contained. If a project spans more than one file, an identifying line for that project must be placed in the control procedure for each file to which the project applies.

While adding, deleting, changing and commenting out project identifier are all simple tasks, the user should keep in mind that these identifiers are control nodes for applying and backing out whole sets of updates. Simple actions taken on these nodes can have major consequences. Of particular importance are the following issues:

Order: Updates are applied in the order shown in the Control procedure. A project, being the name of a logical change, may point to many update procedures (physical changes). If the order of project names is altered in the control procedure, and update procedures across projects affect the same section of the same procedure, the sequence numbers in later updates may not match those in the existing procedure, and the attempt to reconfigure the file will fail. In this case the ReConfigure option in *SirLib* will tell the user the update procedure that failed to apply and the sequence number where the error occurred.

Logical Dependency: There is no way *SirLib* or any other change management system can know about logical dependencies between source code changes in different update procedures. A %variable declared in one update may be used by code built by another update procedure. Backing out the first change may allow the second to apply correctly (that is, all the lines of code physically verify before they are applied). But the resulting program may not function correctly because the declaration of the %variable is missing from the resulting procedure. It is important that physical changes which are logically dependent upon on another be applied and backed out together, and this is done by linking them to the same project.

From the Project Definition screen, PF2 gives access to an edit session in which the user may add as much documentation as required to the project. The user edits into a procedure in mixed-case mode, and when the edit session is exited, the lines of the procedure are converted to TEXT fields in SIRLIBD, and linked to the appropriate Project Identifier.

6.2 Configuring Files (Applying Updates)

Applying updates is always done a *file* at a time. The process is called ReConfiguring a file. The file and FixFile are selected on the main menu at the time Option 2 is selected. The SIrLib File Reconfiguration screen is then displayed:

```
------------------- * * * SIRLIB File ReConfiguration * * * ----------------

Name of procedure file to be fixed ==> SIRMON
File for CONTROL and change decks  ==> SIRLIBP

This function applies managed updates (change decks) to a Model 204
procedure file.  The control and change decks must be in same procedure
file (SIRLIBP is the system default, but the user may change it.)

The procedure CONTROL.pfile controls file reconfiguration, where 'pfile'
is the name of the file to which changes are being applied.  CONTROL
procedures are maintained via the CHANGES screen in SIRLIB, and contain
8-char change identifiers, followed by an optional asterix and comment.
Change identifiers beginning with '*' are not applied.

Each uncommented change identifier must match one or more procedures
named pfile.fixname.procname, residing in the same file, where pfile
is the file targeted for reconfiguration, fixname is the logical change
identifier and procname is the procedure to which the change(s) applies.

Press ENTER to reconfigure the selected file.
--------------------------------------------------------------------------------
                    PF3/ Exit
```

Applying Fixes/Changes

The above screen shows how the programmer applies changes to a file. Pressing ENTER on this screen applies all active updates in the FixFile to the BASE. version of target procedures in the target file. This option may be protected by the System Administrator, so that only approved users may reconfigure a file.

If a procedure is locked when *SirLib* attempts to apply changes to it, the ReConfiguration fails and a message is displayed identifying the procedure that could not be updated by *SirLib*. This could happen if a user is editing the procedure, or if the procedure is locked by an active APSY. The ReConfiguration should be done over again. ReConfiguration may be run as many times as needed, until all changes are properly applied. When a ReConfiguration fails in this manner the file may be left in a state where some procedures are deleted. The faulty update should be fixed or commented out and reconfiguration run again.

6.2.1 Backing Out Changes

ReConfiguration ignores all projects that are prefixed by an “*”. The manager who wishes to backout all updates linked to a project, can do so by commenting out the project name in the Project screen, and reconfiguring the file using option 2. This takes only a few minutes work and is independent of DUMP/RESTORE or other external backups of the file. To backout only one or some of several updates associated with a project, the manager may delete the update procedure or change its name so it is no longer found when *SirLib* checks for updates linked to projects. Deleting update

procedures is a severe remedy in any case and should be avoided, as the source working procedure from which the update was generated may no longer exist.

Each active Project in the Control procedure must be linked to at least one update, or the ReConfiguration will fail. If updates are deleted or renamed such that none link to a particular project, the project name should also be commented out or deleted.

6.2.2 Applying Updates in Batch Mode

Updates can also be applied in batch mode. The command for applying fixes in batch mode is:

```
SIRLIB BATCH <target file name> <fixfile name> <source file
name>
```

where *<target file name>* is the file to be reconfigured, i.e. needs updating, and *<fixfile name>* is the file that contains the update procedures.

<source file name> is optional and allows fixes originally written against one procedure file to be applied to a different file. This is useful if the source procedure file in the development environment has a different name than the procedure file in production. In the standard case, where the target and source procedure files are the same or have the same name, this parameter can either be left blank or can be the same as the target file.

```
SIRLIB BATCH SIRMON SIRFIXES SIRMON
```

results in the same changes being applied to SIRMON as:

```
SIRLIB BATCH SIRMON SIRFIXES
```

The following example shows changes, originally written against DEVPPROC being applied to PRODPROC (with the change decks coming from SIRLIBD).

```
SIRLIB BATCH PRODPROC SIRLIBD DEVPROC
```

This command does not have to be placed in a batch job, but can be typed directly in at *Model 204* command level, or can be inserted in the USER0 stream, or invoked in an IODEV3 thread.

One use of the batch *SirLib* command is to place the command for each managed file in the USER0 stream of the production online. This allows application updates to be “staged” to production by copying them to the production FixFile, with the application update occuring automatically each time the online comes up.

To aid in batch debugging and job control, SIRLIB also sets a global SIRLIBFAILURE to 0 if the configuration command worked correctly and to 1 if the command failed. This global can be used to set JOBCODEs or to initiate recovery steps in the event the batch

configuration fails. A message detailing the reason for the configuration failure is printed to CCAPRINT.

6.3 Administering System and File Profiles

```
------------------- * * * SIRLIB Administration * * * --------------------

==>
     File:                        SIRMON
     Changes File:                SIRLIBP
     Single Checkout:             N
     Overwrite protection:        N
     Require Change Identifiers:  Y
     Keep History:                Y
     Secure:                      N

     Administrator ID:s           HOMER_____ BART______ MAGGIE____ MARGE_____
                                  ALEX _____ ITCHY_____ SCRATCHY__ __________
                                  __________ __________ __________ __________
     Procedure Stamping:          Y
     Exempt Prefixes:             HELP._____________________________
                                  MSGS______________________________
                                  __________________________________
                                  __________________________________

------------------------------------------------------------------------------
1/Help           3/Quit
7/Up     8/Down  9/Repeat  10/Remove                12/Commit
```

SirLib Administration screen

The *SirLib* Administration Option allows users to define a custom *SirLib* profile for each file. If the Administration option is selected with no file specified on the main menu, then the System Profile is being defined or changed. If a File *is* specified on the main menu then a file-specific profile is being defined or changed.

Only users defined to the ADMIN SCLASS in the *SirLib* APSY definition are allowed access to the Administration Option without specifying a file. Only users defined as Administrators in the System profile are allowed access to File profiles.

The settings on the Administration screen are described below. If any of these settings is left blank on a File profile, then *SirLib* takes its behavior from the System profile setting. If the option is left blank on the System profile, default action varies, but is noted in the descriptions below.

File

A file name is displayed only if one was specified on the main menu. The file name cannot be modified on the Administration panel. If no file was specified on the main menu, then this field will read "*System Defaults*".

Changes File.

A Changes File (FixFile) is a *Model 204* procedure file that contains update procedures and a CONTROL.<filename> procedure for a file or set of files. Any number of files may share the same FixFile. The default FixFile for all of *SirLib* is **SIRLIBP**, a *Model 204* procedure file that was created as part of the *SirLib* installation process. If a different FixFile is specified as the default for a managed file, *SirLib* options 1, 2, 5, and all *SirLib* reports will look in the new FixFile for update procedures and the control procedure.

Single Checkout

This switch affects the behavior of the "Q" command in *SirPro*. The default, *N*, allows multiple programmers to update the same procedure at the same time, and leaves it up to *SirLib* to handle possible collisions when the update procedures are applied. The administrator may switch this option to *Y* to force a "checkout" to occur when the "Q" command is executed against a procedure.

Once a procedure is checked out by a programmer it may not be checked out by another via a managed update command until the previous programmer signals that it is available by checking it back in. The checkout occurs automatically when a "Q" command is executed. The checkin is optional on the "X" screen, when the final update procedure is being generated. The programmer simply checks "Y" at the bottom of the Xcompare panel to unlock the procedure. Users in the *SirLib* ADMIN sclass can view and remove procedure locks by Option 7 from the *SirLib* main menu.

Regardless of the setting of this switch, *SirLib* stores a *lock* record for each procedure that is the subject of a "Q" (Sequence) command. This allows managers to view a list of procedures being updated even when the procedure is not actually locked.

Overwrite protection

This switch affects the behavior of the *SirPro* "X" prefix command. By default *SirPro* allows users to overwrite each others' update procedures, provided a switch is checked on the Xcompare panel when generating the second update procedure. The Overwrite switch on the Administration panel never stops programmers from overwriting their own changes, however if Overwrite is set to *N* on this panel, *SirLib* will not allow a programmer to replace an update procedure generated by another user.

Require Project Identifiers

The system default is to allow programmers to generate update procedures for any project, whether or not the project name is listed in the *CONTROL.<filename>* procedure for the file. If this toggle is set to *Y*, then programmers may only name their update procedures using project names that exist in the control procedure. Note the format of update procedure names is:

```
<filename>.<project>.<proc-name>
```

The second qualifier of the above naming convention is the value that is checked when this option is set to *Y*.

Requiring project names to exist before updates are linked to them allows the manager to specify destinations for update procedures, and may prevent changes from being left out of a release by inadvertent misspellings of project names by programmers.

The advantage of *not* requiring a project name to exist before changes are linked to it is that it provides a "control point" to switch on an entire project. That is, update procedures may be generated to FixFiles over a period of time, but they don't get included in file reconfigurations until a project name is entered, and at that point they are all made active simultaneously.

Keep History

Setting this switch to *N* reduces the amount of historical information kept by *SirLib*. Certain history records are maintained regardless of this setting, such as ReConfigurations and Cutovers.

Secure

This option globally turns on and off *SirLib* internal security, for the system as a whole or for a particular file.

If this switch is set to *N* the only security in effect is determined by users' SCLASS privileges from SUBSYSMGMT. ADMIN SCLASS users only are permitted access to the Administration and View/Clear Procedure Locks options from the main menu, and all other options are available to any user with access to the *SirLib* subsystem. If SECURE is set to *Y*, then *SirLib* will pay attention to the values set by administrators in the SECURE option.

If on the default System profile, SECURE=N, specific files that require security can override this setting by having SECURE=Y on their administration screen. The reverse can also be done, with the system set to use security by default, and files that need no such detailed coverage set with security off. A blank setting for SECURE on a file record tells *SirLib* to use the System Defaults for that file. If SECURE is left blank in the system profile *SirLib* assumes it should be *N*.

Administrator IDs

Administrator IDs defined for the System profile (i.e. when no file is specified on entering the Administration Option) are given access to Administration functions for all files. These System Administrators may then access Administration options to set File profiles. Users defined as Administrators for a File have access to Security (Option 5) for the files they administer. Up to 12 IDs may be specified as Administrators for the system and for each file. If a shop wishes to use *SirLib* and *SirPro* on an honor system basis, one of the simplest ways to do it is to define all users to the ADMIN SCLASS of *SirLib*, and either turn security off, or let each user build whatever permissions they require for themselves or their development group.

Procedure Stamping

This switch indicates whether or not a comment is inserted in a changed procedure. The comment indicates the name, date, time, and source file of the change. Procedure Stamping may be set on or off globally (*Y* to turn stamping on, *N* to turn it off) on the Administrator setup screen.

Internal procedure stamps can be deleted by *SirLib* when the file is "Cutover" (refer to "Release Cutover" on page 42). The deleted stamps may be replaced with

comment stamps indicating the cutover date, time, and administrator ID. Default *SirLib* behavior is to stamp procedures.

Note: When working versions of procedures are generated (via the "Q" prefix command in *SirPro*), internal stamps are stripped from the working procedure and sequenced procedure. This prevents programmers from having to deal with these extra comment lines, and it prevents the stamps being generated into update procedures. Procedure stamps are also used as input data in some *SirLib* reports.

Exempt Prefixes

If procedure stamping is "on", a list of procedure prefixes may be entered for procedures in the system or for a file which should not have comments inserted. This prevents *SirLib* comments from entering procedures that contain application help text, message text or other literal information used by the application. Prefixes for exempt procedures may be up to 32 characters long. Wild card characters such as "*" or "?" are taken as literal values in these prefixes. Up to 4 prefixes may be specified for any file or the system as a whole. There are no default exempt prefixes.

Files are added to the *SirLib* system via the Administration panel. For a file to participate in Managed update activity, it must have at least a file name entered on this panel, and the entering user must have pressed

To remove a file from the *SirLib* system, simply press PF10 and *SirLib* will remove all security, history, procedure locks, administration and user privileges for the file. If the file has already been added to the *SirLib* subsystem, use SUBSYSMGMT to remove it.

6.4 Release Cutover

One way to use *SirLib* is to keep update procedures forever, allowing unlimited flexibility to reconfigure a system to a previous state. However, as systems age it may be inconvenient to keep applying a large number of projects and even larger number of update procedures. The “cutover” option essentially returns a managed file to a pre-managed state, but with all existing updates applied.

Cutovers always occur on a file-by-file basis. Performing a cutover causes the following *SirLib* processing:

ReConfiguration: Cutover invokes a full file ReConfiguration to ensure that all updates are applied. This step can be bypassed with a switch on the Cutover panel.

Delete Update procedures: All update procedures in the FixFile that have the cutover file as a first qualifier are deleted whether or not they are linked to an active project.

Delete Project Identifiers: All lines in the control procedure for the file (CONTROL.<*cutover-file*>) are deleted.

Delete BASE. Procedures:: All procedures beginning with “BASE.” are deleted from the target cutover file.

Delete Comments: When changes have been applied, *SirLib* will have inserted a comment like:

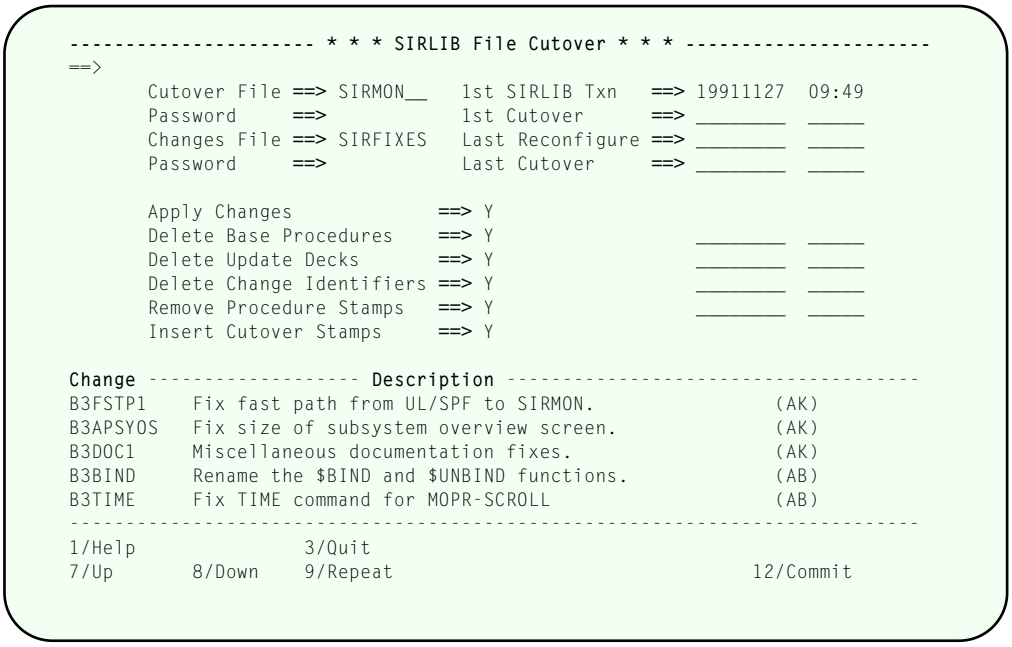
```
*** F2BALES Applied by ... ***
```

at the beginning of each changed procedure. These comments are deleted at cutover, and a single new comment reading:

```
*** Cutover on DD/MM/YY
```

is inserted (procedures excluded from inserted commenting are not stamped; see System Setup). The re-stamping of procedures with a Cutover stamp can be suppressed by a switch on the Cutover screen.

Cutover is initiated from the *SirLib* main menu, via option 4. The file to be cutover is specified on the main menu along with the FixFile from which the changes are to be applied and deleted. If the FixFile field is left blank the default FixFile from the File record is used.



Cutover Information Display

A scrollable panel at the bottom of Cutover screen allows the user to view the changes that are to be applied. If the projects shown are not the ones the user wants applied and deleted, the process should not be continued until the appropriate FixFile is located.

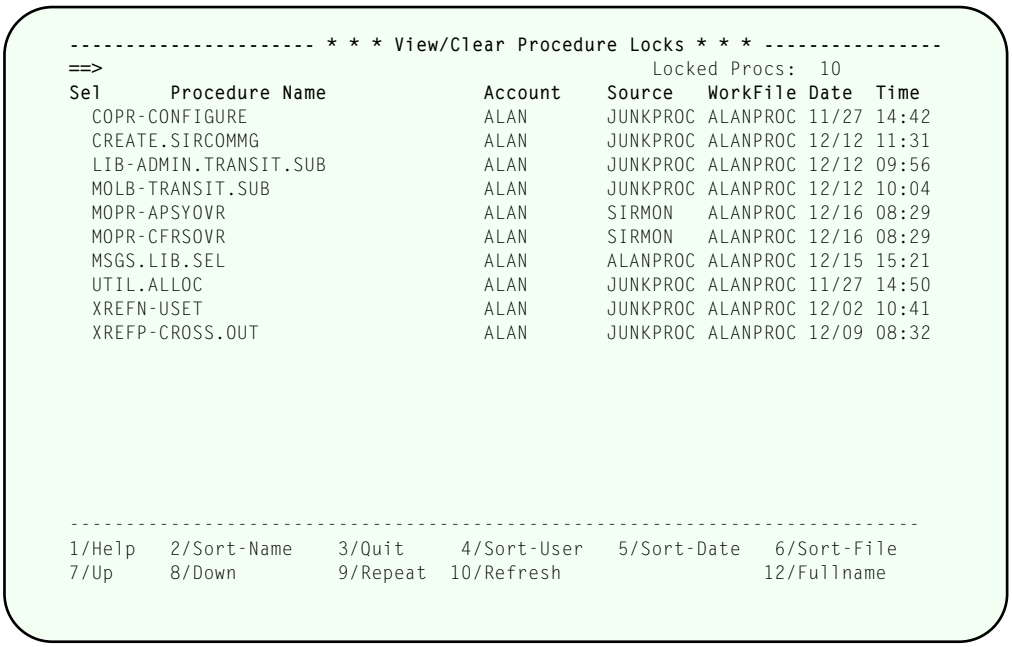
The remaining fields on the Cutover screen display information to help the file manager determine the state of the file and changes associated with the file. Existing projects are displayed in a block at the bottom of the screen. Dates and times are displayed for the first *SirLib* transaction for this file, the first and last reconfiguration, and the most recent cutover. All of this information is refreshed after a successful cutover.

From the Cutover panel, PF12 initiates the cutover. If the cutover fails during any step a failure message appears when the screen is refreshed, and the user should check the date and time stamps for each activity to see which step failed. Refer to “Problem Resolution” on page 53 for advice on error recovery.

Because the results of a cutover are irreversible it is recommended that cutovers always be prefaced by a backup of the managed file and its associated FixFile. It is also recommended that the Reconfiguration step not be bypassed unless the user knows for certain that a reconfiguration has just been run. Care should be taken to verify that changes are not incorrectly “commented out” in the Control procedure. If any step fails to execute properly, processing stops at that step and control is returned to the Cutover panel, where an error message is displayed. After cutover compleyes, the date stamps for each activity will be updated on the Cutover panel. The user should verify these dates and times before exiting.

6.5 View/Clear Procedure Locks

Procedures always have a record posted whenever a programmer takes a managed update copy of the procedure using the “Q” command. This record becomes a lock when the manager has set Single User Update to *Y* in the Administration panel. Programmers unlock procedures by executing an “X” (Xcompare) against the locked procedure, specifying *Y* in the field on the Xcompare screen that asks whether the procedure should be unlocked. The View/Clear locks option in *SirLib* is a way for managers to track and modify procedure locking.



View/Clear Procedure Locks

Access to this option is determined by ADMIN SCLASS membership.

If the manager wishes to clear a lock, an “S” is placed to the left of the procedure name. Locks are cleared when ENTER is pressed. If the View/Clear Locks function is entered with a filename specified on the main menu, only those procedures locked from the specified file are displayed. The list may be scrolled and sorted via the displayed PF keys.

In the View/Clear locks option, the procedure name is displayed out to 33 characters. Following that, the display shows the programmer owning the procedure, the source file for the checkout, the workfile and the date and time the checkout occurred. PF12 allows the view to be toggled into a fullname mode, in which long procedure names overlay the user and file information on the right side of the screen.

CHAPTER 7 Security

SirLib security does not directly protect files. Because programmers will be working most of the time in *SirPro*, a simple way to ensure consistent use of the *SirLib*/*SirPro* system is to make all managed procedure files PUBLIC with low (read-only) privileges, and then to allocate those procedure files to *SirPro* with read privileges and to *SirLib* with update privileges. This allows programmers to make copies of the procedures they need but not to change anything in the managed files, forcing all updates to go through *SirLib*.

Details on this sort of conversion are given in “Getting Started” on page 61.

CHAPTER 8 *SirLib Reports*

This section reviews the reports supplied with the *SirLib* system. The Reports menu is accessed as option 6 from the *SirLib* main menu:

```
---------- * * * Configuration and Change Control Report * * * -----------

==> 3
        1.  Change History
        2.  Configuration Rpt         File    ===>
        3.  Problem Tracking          FixFile ===>

   *------------------ Directed Output Specification  ------------------*

   Destination ==> 2 (Select 1, 2 or 3, below)

   1. DATASET DDNAME   ==> OUTALAN
   2. PRINTER ID       ==> ALAN
   3. $PRINT CLASS     ==> X

   Lines Per Page      ==> 60  (UDDLPP)
   Characters per Line ==> 133 (UDDCCC)
   Record Format       ==> 12  (UDDRFM)
   Header Control      ==> 3   (HDRCTL)

--------------------------------------------------------------------------------
1/Help           3/Quit
                                                                  12/Submit
```

SirLib Reports Menu

Reports are selected by number. Reports may be limited to a File and/or FixFile. Print routing and formatting information is entered in the fields at the bottom of the screen. A print destination of TERMINAL, $TERMINAL or blank will result in the report printing at the user's terminal. If a printer is specified, the printing activity is handled by an SDEAMON and the user's terminal session is freed immediately upon their pressing PF12.

CHAPTER 9 Configuration Options

Manipulating update procedures instead of whole procedure files offers a large number of advantages in managing the *Model 204* environment. These advantages are seen in simplified and more flexible pathing schemes for promoting changes, clearer accountability for change, and a more powerful and direct way of building a bug-free production system.

The flexibility of *SirLib* allows sites to use the product in a number of different ways. The best *SirLib* implementation is the simplest one that handles all the complexities of a site's application needs. Factors to consider include:

- The number of regions available for developing, testing, maintaining and running production *Model 204* systems.
- The relative volume of new development vs. maintenance activity.
- The complexity and length of application testing.
- Local coding standards.

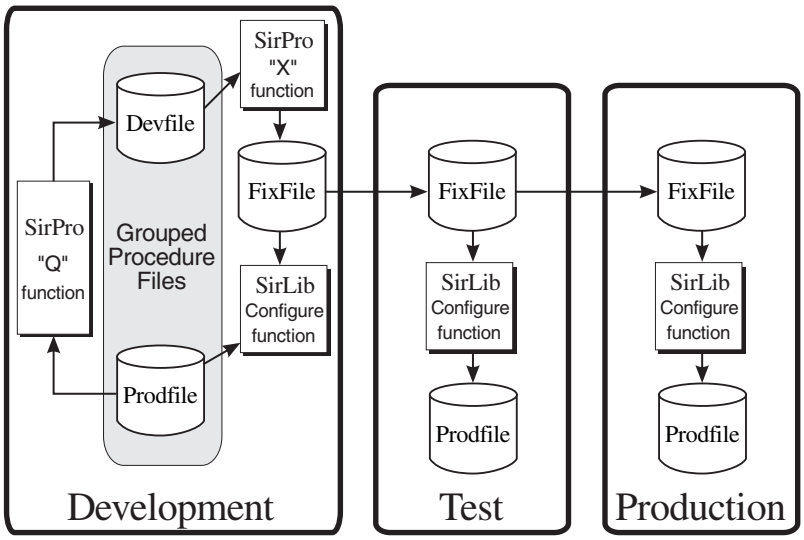
In general, shops want to accomplish as much as possible in each available online region. Therefore, functional areas tend to be merged whenever possible: either development and testing or development and maintenance are run in a single region, with some shops having the luxury of a unique region for each functional area, and other shops having to develop and test, run and maintain production all in a single region. *SirLib* can accomodate any of these situations, but here we would like to present a kind of optimal situation, where functional areas are merged into a single region whenever it is possible without causing programmers, testers and users to interfere with each other.

A practical limit with *Model 204* makes it difficult to merge more than two functional areas into a single region. Assuming that a shop is taking advantage of the Application Subsystem (APSY) feature of *Model 204*, and that there are embedded references to procedure files in the User Language code (e.g. statements like “IN PROCFILE INCLUDE xxxxxxxxx”), only two versions of any application subsystem can be run in the region -- one with the specified procedure file, and a second whose APSY definition has a procedure group defined with the same name.

If a shop wants to run three versions of a subsystem in a region, say a development APSY, a test APSY and a copy of the production APSY, programmers either have to change procedure references when moving procedures between subsystems, code in such a way that no procedure file references are used, dynamically free and allocate files, or change the definition of the procedure group used by one of the apsys. A shop might also code all procedure file references with dummy string substitution or, as

mentioned above, not use the Application Subsystem feature. Any of these options sacrifices some of the strengths of *Model 204*, or places unnecessary restrictions on coding staff.

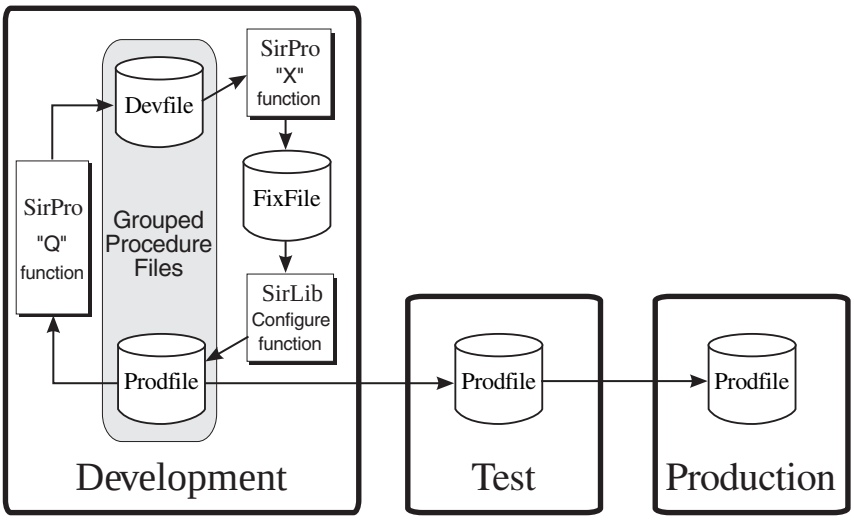
Given that no more than two functional areas should operate within a region, and assuming a shop requires a full-scale testing environment, the following example shows one possible implementation of change management using *SirLib*.



Managing changes across 3 environments, using SirLib in each

The problem with the above example is that any volume of maintenance fixes will interfere with the normal course of development. With testing given its own online region, programmers will have to balance scheduled development with production fixes. Development may have to be set aside (by temporarily renaming procedures being worked on, for instance) while emergency fixes are coded, then the fixes will have to be incorporated with the new development work quickly, as the region is turned back over to development tasks. If fixes to production are a regular occurence this can be an expensive and time-wasting way to work.

If a shop wants to eliminate the complexity of dealing with update procedures in every region, the previous flow of change could be simplified to look like this.

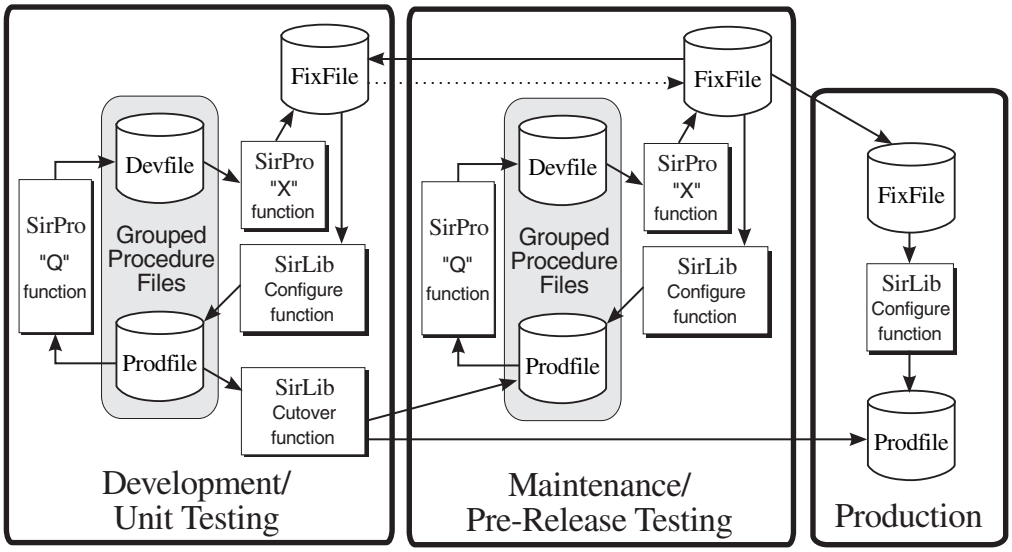


Managing changes across three environments, using one copy of SirLib

Again, this configuration has the same problems with emergency fixes and other unscheduled maintenance interfering with the development schedule, and the ensuing complications in the development environment. In addition, while you gain the simplicity of only dealing with update procedures in the development region, you lose some of the strengths of *SirLib*: the ability to verify and reconfigure the state of procedure files in any region and the simplified distribution provided by the FixFile. In addition, under this scheme, testers need to be given the entire test file over again every time changes are requested of the development team, and of course, the entire procedure file must be sent to production each time a release is made.

A final example of a *SirLib* configuration is the way that Sirius Software used the product for its own development and maintenance distribution.

Because Sirius' “production” environment was a large number of client sites, and the amount of maintenance required on a release was unpredicatable, Sirius used a single environment for development and testing, and it reserved a second environment for ongoing maintenance on each production release. This configuration allowed the building of new releases in the same Online region where complex testing occurred, and the simultaneously re-configuring of the maintenance environment to match that of any client experiencing problems. An example of this setup is shown below:



Separately managed development/unit testing and release testing

The “next” release of an application was developed in the Development/Testing region, using *SirLib* to manage the staging of completed units.

This is an excellent option for anyone with a large volume of development work in progress, complex testing requirements, and production environments that may require unsheduled programming not easily integrated with ongoing new development.

CHAPTER 10 *Problem Resolution*

This section reviews corrective actions for a variety of common error conditions.

10.1 SirLib Can't Access File

Make sure that each file managed by *SirLib* is allocated via SUBSYSMGMT to the *SirLib* APSY, or that an update password exists for the file.

10.2 Cutover Failed

Cutover is a complex function involving 5 steps:

1. Apply updates,
2. delete update procedures,
3. delete project names,
4. delete “BASE.” Procedures and
5. ReStamp procedures.

Only the first and last options may be bypassed. Each of the 5 options is executed in the above order. If a step fails it can be identified by the error message and by the date/time fields associated with it not having been updated when control returns to the Cutover screen.

The first step applies update procedures to the target managed file. This process is exactly the activity performed from the Reconfiguration option, and may be bypassed if the user has just run a Reconfiguration. If the first step is executed (recommended always) and fails, then the destination procedure file does not have all existing changes applied to it. This change application process *does not* apply changes which are commented-out in the control procedure, and it cannot know if all update procedures that are supposed to be in the FixFile are indeed there. Most of the error messages occurring at this step will indicate that some user or apsy is locking a procedure that *SirLib* wants to update. See the ReConfiguration section for details on this.

Each of the remaining 4 steps is an updating transaction requiring update privileges to either the update procedure file or target procedure file. The first thing to check in case of cutover failure is that these files are allocated to the online region and to the *SirLib* APSY, and that privilege settings are high enough to allow the updating of permanent procedures.

If cutover continues to fail, either turn on all messages in the *SirLib* ASPY definition, or run *SirLib* in TEST DEBUG mode, and call the specific problem in to your *SirLib* Administrator or to Technical Support. Cutover may be run as many times as necessary to accomplish the file clean up.

10.3 Missing Base Procedure

If a BASE procedure of a file once existed and has since been deleted, *SirLib* will see there is no BASE, copy the existing executable procedure into a BASE version and then attempt to apply changes to it. The user will get an error saying that changes (from the first update procedure most likely) will not apply to the procedure. The error message will specify the sequence number where the problem occurred. The user will be able to edit both the procedure and the update procedure and see that the new code in the update procedure being applied already exists in the BASE procedure.

A BASE version of a procedure is created the first time a change to the procedure is detected during a reconfiguration. If a BASE procedure is not present when a re-configuration is attempted, *SirLib* assumes this is the first time managed updates have been applied to the procedure, and a new BASE. procedure will be created.

The BASE procedure is more important to a managed system than the actual executable procedure. BASE procedures with update procedures applied are used to re-build the executable procedures to any previous configuration.

If a BASE procedure is accidentally deleted while there are still changes that need to be applied to that procedure, the BASE procedure *must* be restored for those changes to apply. Each BASE procedure must be in the same file as the executable procedure which it matches.

10.4 Missing SEQ Procedure

Sequenced versions of procedures are generated during execution of the “Q” command. The SEQ. version created must remain unchanged while the unsequenced (working) copy is being updated. If the SEQ. version is altered, the update procedure resulting from the XCOMPARE will not apply correctly to the BASE procedure, and the entire reconfiguration will fail.

If an SEQ. procedure is deleted or is suspected of being altered, the programmer needs to generate a new SEQ. version via the following steps:

1. Verify that no unmanaged changes have occurred against the procedure since the working version was generated.

2. If managed changes have occurred against the procedure since the original sequenced version was generated, the user should comment them out in the CONTROL procedure via the CHANGES option in *SirLib* and regenerate a new SEQ version that includes those changes, by executing a “Q” command.

3. Regenerate the sequenced and unsequenced procedures from the original using the Q command, sending the unsequenced version to a temp file, which can be deleted.

10.5 Deleted Update Procedure

Update procedures can be regenerated as many times as required, as long as the original sequenced and unsequenced versions of the procedure still exist.

10.6 Reconfiguration Failed

Usually Reconfiguration failures are due to an APSY or USER holding the target file open. Reconfiguration requires exclusive access to the target file. *SirLib* does not check that a specific APSY is locking a file, as it makes no assumptions about how many APSYs might be running out of that file. Therefore, *SirLib* always presents this error upon its inability to delete or rename some procedure. Reconfiguration can be restarted when the locking USER or APSY releases the file.

If an invalid update procedure has been built and reconfiguration fails, a message is displayed indicating the update procedure at fault and the sequence number where the mismatch occurred. Reconfiguration will also fail in the case where two or more programmers have attempted to change the same line number from working and SEQ procedures generated at the same time. In *SirLib* this is known as a “collision”.

10.7 Collisions

Update collisions occur during ReConfiguration when two update procedures attempt to change the same line or section of a procedure, both referring to the same physical location in the procedure by the same sequence number(s). This can occur when more than one programmer generates sequenced and unsequenced copies of the same procedure in a time frame that doesn't allow the second programmer's copy to include changes from the first programmer's update procedure. Read the section under the Q command for an explanation of how sequence numbering works. In the case of collisions, the first change gets applied by *SirLib*, and the second change contains commands that try to find sequence numbers that are now altered by the first change.

When *SirLib* applies update procedures to BASE procedures, it sorts all changes for a procedure into sequence and version number order. If *SirLib* finds within a section of code two changes that both want to be the same version it warns the user and stops processing.

There are many ways to handle collisions and the best is usually determined by the programmers. A simple approach is to delete all but the most complex of the conflicting update procedures and to incorporate the other changes into the working version of the procedure that was used to generate the one update procedure kept. That update procedure is then regenerated. In some cases the working and SEQ version of the most complex update procedure will have been deleted. In this case it is best to regenerate the conflicting update procedure using an SEQ procedure that contains the other change(s).

To avoid difficult resolution of collisions, working and SEQ procedures should never be deleted until the programmer has verified that the changes apply and integrate with all other changes.

10.8 Production Fixes

Sometimes it will be necessary to make a quick change to a production procedure which has already been updated in development. In *SirLib* developers should never copy production procedures back to development in order to make these fixes.

In the production environment identify which changes have been applied to the procedure which requires the fix. This can be done by browsing the procedure to see the change stamps, or if procedure stamping is turned off for the target procedure, by running a status report for the file to identify the last Reconfigure date and time, and the projects which were then applied.

In the development environment, ReConfigure the file to its current production state. Execute a “Q” command to generate SEQ and working versions of the procedure. Make the production fix, generate an update procedure and distribute the new update procedure to production, applying the update procedure there via a ReConfigure.

In the development environment, ReConfigure the file to include the new fix and all other new development update procedures. Handle collisions as indicated earlier in this section. Notify all developers working on the fixed procedure to regenerate their SEQ and working procedures to include the production fix.

APPENDIX A SIRLIBD Record Structure

SirLib stores a variety of change management information in the *Model 204* file SIRLIBD. This data is used in the *SirLib* reporting system (Option 6 from the main menu) and may also be processed by user-provided reporting programs. This appendix explains the record layout for SIRLIBD.

RECTYPE Field indicating record type, with following values:

- **SYS** Rectype that specifies system defaults for *SirLib*. There is only one record with this RECTYPE.
- **FIL** Rectype that specifies file defaults for *SirLib* (missing fields on this record means use system default). One record of this RECTYPE for each registered file.
- **PRO** Unused.
- **USR** Rectype for user security definition.
- **DOC** Documentation associated with a project.
- **CHG** Project Identifier record.
- **PRC** Procedure record for procedures checked out by programmers.
- **HST** Historical information kept if “Keep History” is specified in file or system defaults.

CHANGE Project name in CONTROL.<file>.

FILE Name of managed update file.

FFILE Name of update procedure file.

PROC Procedure name.

PROJECT Unused.

RESOURCE Name of a protected resource, as follows:

- **CHANGE** Access to Change Definition.
- **CONFIG** Access to Re-Configuration.

CUT Access to File Cutover.

RESEQ Access to Procedure Resequencing -- “Z” prefix command in *SirPro*.

SECURE Access to File-specific security.

USER User ID (from $ACCT).

WFILE Work File. File that procedures were moved to via one of the managed update commands.

ACTIVITY The managed update activity tracked on a history record (RECTYPE = HST).

ADMIN Administrator id specified on the SYS rectype.

EXEMPT Prefix of procedures not to be stamped with internal comments on the status of the application of changes. Maximum length is forty characters.

DATE General date/time stamp, in *YYYYMMDDHHMM* form.

HISTORY Single character flag indicating whether historical information should be kept for a file or system. Contains either blank, Y (yes) or N (no). If RECTYPE=FIL, blank means refer to the setting on RECTYPE=SYS. If RECTYPE=SYS, blank means *keep history*.

OVERWRITE Single character flag indicating whether programmers are allowed to overwrite each other's update procedures (overwriting one's own update procedure is always allowed, though a switch on the XCOMPARE screen will help prevent accidental overwrites). Contains either a blank, Y (yes) or N (no). If RECTYPE=FIL, blank means refer to the setting on RECTYPE=SYS. If RECTYPE=SYS, blank means *allow overwriting*.

PROC.LOCK Single character flag indicating whether procedures should be locked against managed updates once a developer has checked out the procedure (checkout occurs when a “Q” command is executed against the procedure). Contains a blank, Y (yes) or N (no). If RECTYPE=FIL, blank means refer to the setting on RECTYPE=SYS. If RECTYPE=SYS, blank means *don't lock procedures*.

REQ.IDENT Single character flag indicating whether an existing project identifier is required for generating update procedures. Contains a blank, Y (yes) or N (no). If RECTYPE=FIL, blank means refer to the setting on RECTYPE=SYS. If RECTYPE=SYS, blank means *don't require identifiers*.

SECURE Single character flag indicating whether the SECURITY settings defined in Option 4 of the *SirLib* main screen should be enforced. Contains a blank, Y (yes) or N (no). If RECTYPE=FIL, blank means refer to the setting on RECTYPE=SYS. If RECTYPE=SYS, blank means *don't secure*.

STAMP Single character flag indicating whether managed procedures should be stamped with a single line comment for each change that has been applied. If RECTYPE=FIL, blank means refer to the setting on RECTYPE=SYS. If RECTYPE=SYS, blank indicates that stamping *should be done*.

TEXT Text for change comments and documentation on RECTYPE=DOC. Maximum length is 72 characters.

TIME Time stamp, in *HHMM* form.

CUT.BASE Date/Time stamp from the most recent cutover of BASE procedures (RECTYPE=FIL), in *YYYYMMDDHHMM* form. When BASE procedures are cutover they are deleted.

CUT.CHANGE Date/Time stamp for the last time Update procedures were cutover (RECTYPE=FIL), in *YYYYMMDDHHMM* form. When Update procedures are cutover they are deleted.

CUT.IDENT Date/Time stamp for the last time Project Identifiers were cutover (RECTYPE=FIL), in *YYYYMMDDHHMM* form. Project Identifiers are deleted from the "CONTROL.*filename*" file when they are cutover.

CUT.STAMP Date/Time stamp for the last time internal procedure stamps were cutover, that is, removed from procedures in the file (RECTYPE=FIL), in *YYYYMMDDHHMM* form.

FIRST.CUT Date/Time stamp for the first cutover performed on a file (RECTYPE=FIL), in *YYYYMMDDHHMM* form.

LAST.CUT Date/Time stamp for the last time Cutover performed on a file (RECTYPE=FIL), in *YYYYMMDDHHMM* form.

LAST.CONF Date/Time stamp for the last time a file was reconfigured (RECTYPE=FIL), in *YYYYMMDDHHMM* form.

UNLOCK Date/Time stamp when a procedure was checked back in by an updating programmer (RECTYPE = PRC), in *YYYYMMDDHHMM* form.

APPENDIX B Getting Started

Following are the steps the *SirLib* administrator should follow before *SirLib* and *SirPro* are made available to programming teams.

1. Follow the instructions in the *Sirius Mods Installation Guide*.

2. Follow the *UL/SPF* installation instructions (**http://m204wiki.rocketsoftware.com/index.php/UL/SPF_installation_guide**).

3. Add users to the *SirLib* and *SirPro* Subsystems. ADMIN SCLASS users in both *SirLib* and *SirPro* acquire STOP/START/TEST privileges. In SIRLIB, ADMIN SCLASS users also have the ability to access the Administration Option which allows them to define file administrators for *SirLib* purposes, and allows them to define or change *SirLib*'s system default profile.

   SIRPRO should generally be left PUBLIC and AUTOSTART.

   SIRLIB may be either PUBLIC or PRIVATE, depending upon how strictly you want to manage project definitions, reconfigurations and access to reports. PRIVATE is the default.

4. Add files to the *SirLib* apsy definition.

   For *SirLib* to manage changes to procedures in a *Model 204* file, the file must be added to the apsy definition via SUBSYSMGMT, with X'BFFF' privileges in all SCLASSs. By making managed files public, and requiring that update access occur only through *SirLib*, the administrator can guarantee a high level of integrity for the state of changes in the file. When files are allocated to the apsy, they should be allocated as ***non-required***.

5. For procedure files to be managed by *SirLib*, change OPENCTL to X'80' and PRIVDEF to X'0221'. This allows read access to managed procedures, and allows *SirPro* to be used without the need for passwords. At the same time it protects procedures from update outside the change management system.

6. Create a *SirLib* default system profile. *SirLib* requires a single “administration” record to exist in its internal data file (SIRLIBD) for the system profile. To build the system profile record, choose Administration from the *SirLib* Main Menu without specifying a file, define the characteristics of your *SirLib* setup, and save the profile. Use the help text or the appropriate section from this manual for advice on what value to set in each screen field.

7. Create a *SirLib* Administration record for each managed file. *SirLib* requires an "administration" record to exist in SIRLIBD for each file to be managed. Add the file-specific Administration record by choosing Administration from the *SirLib* Main Menu, specifying a file name at the FILE prompt. Use the help text or the appropriate section from this manual for advice on what value to set in each screen field. Any field left blank on the file-specific administration screen means that that characteristic will be inherited from the system profile.

   Users will not be able to perform managed update commands in *SirPro* for a particular file until the *SirLib* Administration record is created for the file. *SirPro* managed update commands are X, Q, K, N and Z. All other *SirPro* commands (Edit, Browse, Rename, etc.) will operate independent of *SirLib* Administration records.

8. Define *SirLib* security characteristics. *SirLib* Security is ignored if SECURE is set to "N" on the System Administration record (which you would have built two steps previously). The security scheme is put back into effect when SECURE is switched back to "Y" in Option 3. Setting SECURE=N is a good way to let users navigate freely around *SirLib* while they are learning the system.

   To define *SirLib* internal security, select Security from the *SirLib* Main Menu, not specifying a file at the FILE prompt. The top line of the Security screen shows system default security settings, and subsequent lines show settings for each file that has an Administration record. You protect a *SirLib* function on a system-wide basis by placing any character in the system default row (the top row of the screen). So, for instance, you could protect every *SirLib* function except Reports, by placing a "Y" in the top row for each column except the last one.

   In the file-specific rows of this screen, a blank column allows the file to inherit the system default setting for that *SirLib* function, a "Y" means protect the function regardless of system default setting, and a "N" means do not protect the function regardless of system default setting. This scheme allows you to define a system default security scheme, and then let all files inherit the same scheme.

   When a function is protected for a file, individual user ids must be given permission to access that function on a file by file basis. To give users this access, select Security from the *SirLib* Main Menu, specifying the procedure file at the FILE prompt. The Security screen will again appear, but this time the top row represents the settings for the file-specific functions, and each subsequent row allows the entry of a user ID.

   On the file specific security screen, you cannot alter the top row (use the system default screen to change file protection). Enter user ids in the rows that follow, and place "Y" in the columns for the protected functions you wish that user to be able to access.

   Again, if you start out using *SirLib* with SECURE set to "N" in the system default Administration record, all internal security is bypassed. You can still build your security scheme, but users are not restricted by it.

A number of other changes can be made if you wish to make the *SirLib*/*SirPro* controlled environment more convenient for programmers. The following changes/recommendations allow programmers to work in GROUP procedure file context, supporting “development” and “production” versions of an apsy in the same region.

9. Allocate a development procedure file for each production procedure file. If you have production procedure files PRCFILE1, PRCFILE2 and PRCFILE3, you might allocate DEVFILE1, DEVFILE2 and DEVFILE3. Under *SirLib*, programmers can never make changes directly in an original procedure file, so the intention of these “DEV” files is to have all changes for PRCFILE1 occur in DEVFILE1, PRCFILE2 in DEVFILE2, etc.

10. Create a GROUP for each production procedure file. The GROUPS should have the same name as the production procedure files. So, continuing the previous example, you would create groups PRCFILE1, PRCFILE2 and PRCFILE3. GROUP PRCFILE1 would be defined as follows:

    ```
    CREATE PERM GROUP PRCFILE1 FROM DEVFILE1, PRCFILE1
    PARAMETER PRCFILE=*, PRIVDEF=x'Ø221'
    END
    ```

    The order of group member definition is critical, as *Model 204* looks for procedures in the order the files are listed.

11. Define a development apsy for each production apsy.

    Use SUBSYSMGMT to copy your production subsystems to development subsystem definitions. So if you have ACCTS, PERSON and INVENT, you might copy each of them to DEVACCTS, DEVPERSON and DEVINVENT. The only change you need to make to the definitions is to change the procedure file to a group.

12. Rationalize your INCLUDE statements. Depending upon how you modularize your User Language code, you may have constructs like:

    ```
    IN FILE ?&PRCFILE INCLUDE COMMON.ROUTINES
    ```

    These statements will fail in GROUP procedure file context unless they are changed to read:

    ```
    IN ?&PRCFILE INCLUDE COMMON.ROUTINES
    ```

13. Build utility apsys for procedure file DUMP, RESTORE and FILEMANAGE. If you've followed the instructions above, your procedure files will now be “PUBLIC” (OPENCTL=x'80'), with very restrictive access privileges. Because the files are PUBLIC, you will never be prompted for a password, so you cannot get access sufficient to perform DUMP, RESTORE or other file maintenance.

The solution is to create apsys that provide the required access, and perform the maintenance for you. These apsy are simple to write, and if you wish, Sirius Software will send you the code for the ones we use, which are DUMPSTER (dump all procedure files), RECOVER (restore a specific file from DUMPSTER dump), and FIMANAGE (customized at each use to perform a specific file maintenance activity).

14. Take a backup of everything you've done.

Finally, *SirLib* users should note the following points.

Baseline Procedures: There is *no need* to establish a baseline for managed procedures. *SirLib* can be put into place to manage existing procedures or to manage systems being built from scratch. The baseline for any file is the point where you begin using *SirLib*. The baseline for any procedure is the state it is in the first time *SirLib* applies a change. The creation and maintenance of a baseline is automatic, and requires no action on the part of developers or managers.

Procedure Prefixes: *SirLib* requires exclusive use of three procedure prefixes. *SirLib* reserves the prefix **"BASE."** for procedures in managed files, the prefix **"SEQ."** for procedures in files where developers are working on changes, and the prefix **"CONTROL."** for update procedures in the FixFile.

APPENDIX C Date Processing

SirLib uses dates in the following ways:

- To examine the CPU clock (as returned by the STCK hardware instruction) to determine the current date, in case *SirLib* is under a rental or trial agreement
- To display the current date, as returned by the TIME SVC, modified by the SYSDATE parameter or the the *Sir2000 User Language Tools* APPDATE clock, as page headers in various end-user displays
- To keep track of the date and time at which a user issued a CHECKIN, CHECKOUT, CONFIGURE, or CUTOVER *SirLib* operation, using the TIME SVC, modified by the SYSDATE parameter or the the *Sir2000 User Language Tools* APPDATE clock

For headers on pages or rows that occur on printed pages or displayed screens, *UL/SPF* products generally use a full four-digit year format, although they may display dates with two-digit years in circumstances where the proper century can be inferred from the context.

Index

S

U

V