AS/400 Software Life Cycle Management with Application Development Manager/400 and SystemView System Manager/400

December 1996

IBM
International Technical Support Organization
Rochester Center
AS/400 Software Life Cycle Management with Application Development Manager/400 and SystemView System Manager/400

December 1996
Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, "Special Notices" on page 269.


This edition applies to Version 3 Release 2, Version 3 Release 6 and Version 3 Release 7 of SystemView System Manager for AS/400 (program number 5763-SM1, 5716-SM1). Application Development ToolSet/400 (program number 5763-PW1, 5716-PW1) features Application Development Manager and Application Dictionary Services for use with the Version 3 Release 2, Version 3 Release 6 and Version 3 Release 7 of OS/400.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1996. All rights reserved.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Contents

Preface ................................................................................................................. ix
How This Redbook Is Organized ........................................................................ ix
The Team That Wrote This Redbook ................................................................. x
Comments Welcome .......................................................................................... xii

Chapter 1. Introduction to the AS/400 Software Life Cycle ............................. 1
1.1 Elements in the Software Cycle ...................................................................... 2
1.2 How to Use and Where to Start ........................................................................ 2
1.3 How to Use the Example Library SG244187.1 ........................................ 3
1.4 Management Elements in the Software Life Cycle ........................................ 3
  1.4.1 Software Development Cycle and Development Process ..................... 3
  1.4.2 IBM Software Tools for the Software Life Cycle ............................... 4
  1.4.3 Software Change Management ......................................................... 5
  1.4.4 Change Management and Impact Analysis ...................................... 6
  1.4.5 Software Packaging, Distribution, and Maintenance ....................... 6
  1.4.6 Problem Management and Corrective Actions ............................... 6
  1.4.7 Support Center/400 ........................................................................... 7
  1.4.8 Project Management .......................................................................... 8
  1.4.9 Managing the Document Control Process .................................... 10
  1.4.10 Managing the Testing Process ....................................................... 10
  1.4.11 Managing the Backup and Recovery Process ............................... 11
  1.4.12 Managing the Software Estimation Process .................................. 11
1.5 Summary ...................................................................................................... 12

Chapter 2. Mailing List Application Description ........................................... 13
2.1 Functional Scenario ....................................................................................... 13
  2.1.1 Inquire Into the Mailing List Master File ........................................ 13
  2.1.2 Maintain Mailing List Master File ................................................... 14
  2.1.3 Submit Mailing by Account Number .............................................. 17
  2.1.4 Submit Special Analysis Report .................................................... 17
  2.1.5 Query Mailing List File .................................................................... 18
2.2 Parts structure .............................................................................................. 18

Chapter 3. Application Development Control Using ADM/400 ................... 21
3.1 Application Development Manager ............................................................. 21
3.2 The Mailing List Application and ADM/400 ............................................ 23
  3.2.1 The Base Project Hierarchy ............................................................. 24
  3.2.2 Importing the Application ............................................................... 24
  3.2.3 Building the Application .................................................................. 26
  3.2.4 Summary .......................................................................................... 30

Chapter 4. Distributing and Supporting Your Application Using ADM/400 ...... 31
4.1 Advantages and Disadvantages .................................................................... 32
  4.1.1 Advantages ...................................................................................... 32
  4.1.2 Disadvantages ................................................................................ 32
4.2 Prerequisites ............................................................................................... 32
  4.2.1 Network Setup ................................................................................ 32
  4.2.2 Install RCVPART Code on All Target Systems ............................ 32
4.3 Exporting Your Application to a List of Systems ....................................... 34
  4.3.1 The SYSTEML Part Type ................................................................ 34
  4.3.2 Create an Installation Program for Your Application (Optional) .... 35

© Copyright IBM Corp. 1996
### Chapter 14. Packaging, Distributing, and Servicing Client/Server Applications

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.1 Packaging Structure of Client/Server Applications</td>
<td>225</td>
</tr>
<tr>
<td>14.2 Distribution of a Client/Server Product</td>
<td>226</td>
</tr>
<tr>
<td>14.2.1 Client Access/400 Update Function, Trigger, and Package Files</td>
<td>226</td>
</tr>
<tr>
<td>14.2.2 Client Access/400 Administration Function</td>
<td>228</td>
</tr>
<tr>
<td>14.3 Sample Scenario Overview</td>
<td>232</td>
</tr>
<tr>
<td>14.3.1 Sample Client/Server Application Package Structure</td>
<td>232</td>
</tr>
<tr>
<td>14.3.2 Packaging the Sample Client/Server Application</td>
<td>237</td>
</tr>
<tr>
<td>14.3.3 Distributing and Installing the Sample Application Client Code</td>
<td>240</td>
</tr>
<tr>
<td>14.3.4 Adding an Exit Program</td>
<td>244</td>
</tr>
<tr>
<td>14.4 Maintaining your Client/Server Product</td>
<td>245</td>
</tr>
<tr>
<td>14.4.1 PTF Client Code</td>
<td>245</td>
</tr>
<tr>
<td>14.4.2 Summary</td>
<td>254</td>
</tr>
</tbody>
</table>

### Appendix A. Diskette Install Instructions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix B. Utility programs</td>
<td>255</td>
</tr>
<tr>
<td>B.1 Making Changes to a PARTL File</td>
<td>257</td>
</tr>
<tr>
<td>B.2 Installing the ADDPART Program</td>
<td>260</td>
</tr>
<tr>
<td>B.3 Using ADDPART Program to Update PARTL File</td>
<td>261</td>
</tr>
<tr>
<td>B.4 Errors That May Occur While Running the ADDPART Program</td>
<td>262</td>
</tr>
<tr>
<td>B.5 Check Out PARTL Parts</td>
<td>263</td>
</tr>
<tr>
<td>B.6 Browsing a Source Part</td>
<td>264</td>
</tr>
<tr>
<td>B.7 Create a CRTPTF Command String</td>
<td>266</td>
</tr>
<tr>
<td>B.8 Display the LICPGM and PTF Information of an Object</td>
<td>268</td>
</tr>
</tbody>
</table>

### Appendix C. Special Notices

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix D. Related Publications</td>
<td>269</td>
</tr>
<tr>
<td>D.1 International Technical Support Organization Publications</td>
<td>271</td>
</tr>
<tr>
<td>D.2 Redbooks on CD-ROMs</td>
<td>271</td>
</tr>
<tr>
<td>D.3 Other Publications</td>
<td>271</td>
</tr>
</tbody>
</table>

### How To Get ITSO Redbooks

- How IBM Employees Can Get ITSO Redbooks                                | 273  |
- How Customers Can Get ITSO Redbooks                                    | 274  |
- IBM Redbook Order Form                                                 | 275  |

### Abbreviations

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>277</td>
</tr>
</tbody>
</table>

### Index

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>279</td>
</tr>
</tbody>
</table>
Preface

This redbook describes the use of Application Development Manager, Application Dictionary Services, and SystemView System Manager/400 in managing, developing, packaging, distributing, installing, and maintaining application software. It introduces these products at the appropriate stages of the software life cycle and gives details of our experiences with these products. The diskette included in this publication contain the sample programs.

This book is intended for customers, IBM Business Partners, independent software vendors, and IBM representatives who need to know or advise others in managing AS/400 applications. A basic understanding of Application Development Manager, Application Dictionary Services, and SystemView System Manager/400 is assumed.

How This Redbook Is Organized

This redbook contains 298 pages. It is organized as follows:

- Chapter 1, “Introduction to the AS/400 Software Life Cycle”
  This chapter introduces some major management elements related to AS/400 software life cycle.

- Chapter 2, “Mailing List Application Description”
  This chapter describes the mailing list application used in this document.

- Chapter 3, “Application Development Control Using ADM/400”
  This chapter describes the use of Application Development Manager in developing the first release of the mailing list application.

- Chapter 4, “Distributing and Supporting Your Application Using ADM/400”
  This chapter describes the use of Application Development Manager in distributing the first release of the mailing list application.

- Chapter 5, “Enhance the Application for National Language Support”
  This chapter describes the use of Application Development Manager in developing an NLS version of the mailing list application.

- Chapter 6, “Package the NLS Release of Your Application Using Application Development Manager and SM/400”
  This chapter describes the use of Application Development Manager in packaging the NLS version of the mailing list application.

- Chapter 7, “Packaging Without the Use of ADM/400”
  This chapter describes the use of SM/400 in packaging a Non-NLS version of the mailing list application.

- Chapter 8, “Servicing Your Products With the AS/400 PTF Process”
  This chapter describes the use of Application Development Manager and SM/400 in developing a simple PTF for the Non-NLS version of the mailing list application.

- Chapter 9, “PTF to Database Files”
This chapter describes the use of Application Development Manager and SM/400 in developing a Database file PTF for the Non-NLS version of the mailing list application.

- Chapter 10, “Developing and Packaging a New Release of Your Application”
  This chapter describes the use of Application Development Manager and SM/400 in developing a new release of the Non-NLS version of the mailing list application.

- Chapter 11, “Supporting Your Product on Previous Releases of OS/400”
  This chapter describes issues involved in supporting your product on previous releases of OS/400. It also deals with some CISC/RISC issues related to packaging.

- Chapter 12, “Additional Packaging and PTF Considerations”
  This chapter describes some additional packaging and PTF considerations.

- Chapter 14, “Packaging, Distributing, and Servicing Client/Server Applications”
  This chapter describes some advanced topics related to packaging, such as using Application Development Manager for ILE programs, doing impact analysis using Application Dictionary Services, product licensing, configuration of SM/400 and using symptom strings support, and First Failure Data Capture for your products.

- Chapter 13, “Advanced topics”
  This chapter discusses how to use the AS/400 system to provide software distribution to attached PCs with PC Support/400 and ManageWare/400.

- Appendix A, “Diskette Install Instructions”
  This appendix describes the process to install the diskette contents onto your systems.

- Appendix B, “Utility programs”
  This appendix describes some utility programs that can be used along with Application Development Manager and SM/400.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Rochester Center.

The authors of the second edition of this redbook are:

Ted Zonderland is an Advisory System Engineer and works in the International Technical Support Organization, Rochester Center. He writes extensively and teaches IBM classes worldwide on all areas of Application Development and Object Orientation. Before joining the ITSO 2 years ago, Ted worked in the Advanced Software Solutions Center in IBM Netherlands.

Marcela Adan is a Senior Programmer and works in the International Technical Support Organization, Rochester Center. She writes extensively and teaches IBM classes worldwide on all areas of System Management. Prior assignments include field System Engineer in IBM Argentina, System Programmer in Nestle Foods Co., as well as working as a developer in the Rochester Programming Laboratory.
Dan Reusche is a Staff Programmer in IBM Rochester. He has 5 years experience in the Rochester AS/400 Production Support department in Rochester I/S where he has implemented an ADM/400 and SM/400 solution for control and distribution of system support tools for his department. He also has system performance responsibilities for the production AS/400 systems used in Rochester. Prior to working in Rochester I/S, he had various assignments in Rochester, Poughkeepsie, and as a Customer Engineer in Moline, Illinois.

Ed Van Weeren is a Senior Advisory System Engineer in the Netherlands. He has over 20 years experience with Rochester products, such as the S/3, S/34, S/38 and AS/400 system. He is currently working in the AS/400 Business Unit with responsibilities for application development. From 1985 until 1988, he worked for the International Technical Support Organization (ITSO) in Rochester Minnesota.

The authors of the first edition of this redbook were:

Marcela Adan
Stephen Lam
Ted Zonderland

The advisor of the first edition of this redbook was:

Jim Cook

Thanks to the following people for their invaluable contributions to this project:

IBM Toronto Development Laboratory:
Lamont Baker
Ted Fines
Doug Mortson
Doug Nybida
Pramod Patel
Brian Leung
Jusni Hilwan

IBM Rochester Development Laboratory:
Mark Anderson
Mike Anderson
Barb Ballard
Nat Calvert
Eric Fosdick
Fernando Herrera
Gerarda Marquez
James Masanz
Ramy Mehta
Mark McKelvey
Randy Rendahl
Kevin Stamschror
Sue Townsend
Sandy Westling

IBM Guadalajara Programming Laboratory:
Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!
Chapter 1. Introduction to the AS/400 Software Life Cycle

This chapter describes all of the elements in the Software Life Cycle. It is important to note that the entire process described in this redbook is not new; it is business as usual for everyone dealing with application creation, maintenance, and distribution. The tools and methods that are discussed help you to structure the process for better control.

If you recognize one of the following problem descriptions as one of yours, continue with the redbook; it assists you in getting better control.

1. Q - Do you ever wonder what the level of implemented software is at multiple sites?
   
   • A1 - Ask the user to display the object information of a program and compare the compile timestamp and source change timestamp.
   
   • A2 - Everyone familiar with the IBM PTF structure knows that you simply ask for the applied PTFs.

   You also can package your application as a product and ask the user the same question - to display the applied PTFs and instantly know the level of the installed software.

2. Q - The year 2000 is approaching fast; are all of your dates six positions?
   
   A - This is just an example of a field attribute that needs to be changed. Application Dictionary Services, the impact analysis tool, can assist you with this task.

3. Q - Did you ever wish that you still had the source from before a change, or if you did keep your source, was it easy to locate?
   
   A - Application Development Manager not only uses multiple levels (such as Production, Test, and Development) for the different stages in the maintenance cycle, but also supports an archive function.

4. Q - Somebody fixed a bug in Release-1. Did anyone tell the developers of Release-2?
   
   A - Application Development Manager provides a notify function; if someone maintains a program in release-1 that has already been enhanced for release-2, the release-2 programmer is notified of the change and can take the proper action.

5. Q - What happened when you applied a fix without knowing that another fix was a prerequisite?
   
   A - With the supersede mechanism in the PTF creation process, this is no longer your concern. All supersedes are recognized and taken care of in the CRTPTF command.

You can probably think of many more problems that you want more control over; this redbook, indeed, discusses many more situations and their solutions.

Other subjects that we want to discuss in this introduction are:

1. Elements in the Software Cycle.
2. How to use and where to start in this redbook.
3. How to use the example library SG244187.1.

1.1 Elements in the Software Cycle

In Figure 1, elements that are part of the software cycle are shown. The intention is not to show the complete picture, but the areas that we want to discuss in this redbook.

```
+--------+ +--------+ +--------+
Require- | SC/400 | | | | |
ments   | | | | | |
+--------+ +--------+ +--------+
Impact   | ADS/400| | | | |
Analysis | | | | | |
+--------+ +--------+ +--------+
Source   | ADM/400| | Others | | |
Control  | | | | | |
+--------+ +---|----+ +--------+
Packag-  | +---V----+ +--------+
ing      | | SM/400 | | |
          | +---|----+ +--------+
+--------+ +---V----+ +--------+
Distri-  | ADM/400| | SM/400 | | |
bution   | RCVPART| | | | |
+--------+ +---|----+ +--------+
          | +---V----+ +--------+
Advanced | +--------+ +--------+
techniques| FFDC | | |
          | etc. | | |
```

Figure 1. Areas in the Development Process

The format of this redbook focuses on implementing a piece at a time. Imagine that the last column of Figure 1 has your current solution for that area. You can concentrate on this area only if you want or need to.

1.2 How to Use and Where to Start

Assume that you have used a 4-GL language to do your impact analysis and source control and you feel comfortable with it. Your distribution of the changes, however, are less advanced. You can start with the packaging of your application as described in Chapter 7, “Packaging Without the Use of ADM/400” on page 71 and the creation of PTFs as described in Chapter 8, “Servicing Your Products With the AS/400 PTF Process” on page 87.

If you feel comfortable with your source control, but want more information on object relationships for the impact analysis area, start with Application Dictionary Services. Although the example described in 13.2, “ADS/400” on page 172 is related to the use of Application Development Manager, you can perform the
same questions for an Application Dictionary Services dictionary build from a product (application) library. That library can be the result of an application development process in either 3-GL or 4-GL. The difference is whether you specify ADM(*YES) or ADM(*NO) on the STRADS command.

If it is your source control that has you up at nights, you may want to focus on the chapters dealing with Application Development Manager.

We suggest that you start with a manageable piece and one that benefits you the most on the short term. Do not try to take on too many pieces at once because multiple small successes are a much better motivator than a large disaster.

1.3 How to Use the Example Library SG244187.1

With this redbook, we ship an example library that enables you to perform all of the examples and exercises in the redbook yourself. After the installation, which is described in Appendix A, “Diskette Install Instructions” on page 255, you can just follow the instructions that are described in the different chapters to use the example code. Besides the example code, we also supply some utility programs that can make life easier for you in the examples as well as in your own implementation of the redbook suggestions.

If you have gone through the examples, try to substitute the example code with your own pilot application code and use the same scenario to guide you in the execution for your own code.

Although the redbook is aimed at V3R7, V3R6 and V3R2, you can load the example library on V3R1 also. All of the examples work on V3R1 as well, unless specified otherwise. In V3R7, V3R6 and V3R2, you can make complete use of TGTRLS(*PRV), which you cannot use for all of the commands in V3R1 or previous releases. In Chapter 11, “Supporting Your Product on Previous Releases of OS/400” on page 153, you find more information regarding the use of previous releases and the consequences.

1.4 Management Elements in the Software Life Cycle

This chapter describes some major management elements related to the AS/400 software life cycle. It introduces the Waterfall and the Spiral software development process models. The positioning of some IBM software tools and how these tools help in managing the software life cycle are described.

1.4.1 Software Development Cycle and Development Process

No matter which development process models you use, they all consist of software life cycle elements of design, coding, and testing as common phases within the models. All of these tools are designed to contribute and are always related to the following most important management factors in software life cycles:

• Software quality
• Schedule of delivery
• Cost of development
1.4.2 IBM Software Tools for the Software Life Cycle

Figure 2 shows the positioning of some of the IBM software tools in relation to the different phases of the software life cycle.

All of these tools can be used independently of any development process model. The Waterfall development process model is used for illustration purposes only.

Note: The tools shown in Figure 2 are not an exhaustive list.

<table>
<thead>
<tr>
<th>PHASE</th>
<th>DEVELOPMENT STAGE</th>
<th>Positioning of some IBM Software Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Initial Business Proposal</td>
<td>/SM600000 /SM600000 /SM600000</td>
</tr>
<tr>
<td>Analysis and Design</td>
<td>Component Level Design</td>
<td>/SM600000</td>
</tr>
<tr>
<td></td>
<td>Module Level Design</td>
<td>/SM600000</td>
</tr>
<tr>
<td>Code/Produce</td>
<td>Coding</td>
<td>/SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000</td>
</tr>
<tr>
<td>Build/Test</td>
<td>Unit Test</td>
<td>/SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000</td>
</tr>
<tr>
<td></td>
<td>Integrated Test</td>
<td>/SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000</td>
</tr>
<tr>
<td></td>
<td>System Verification Test</td>
<td>/SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000</td>
</tr>
<tr>
<td>Production/</td>
<td>Production and Maintenance</td>
<td>/SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000</td>
</tr>
<tr>
<td>Maintenance</td>
<td></td>
<td>/SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000 /SM600000</td>
</tr>
</tbody>
</table>

Figure 2. IBM Software Tools That Can Be Used At Various Stage of Development

<table>
<thead>
<tr>
<th>Tool Symbol</th>
<th>Tool Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM</td>
<td>SystemView System Manager/400</td>
</tr>
<tr>
<td>ADM</td>
<td>Application Development Manager</td>
</tr>
<tr>
<td>ADS</td>
<td>Application Dictionary Services</td>
</tr>
<tr>
<td>CODE</td>
<td>CoOperative Development Environment/400</td>
</tr>
<tr>
<td>WITT</td>
<td>Workstation Interactive Test Tool</td>
</tr>
<tr>
<td>PAS/2</td>
<td>Personal AS/2 Project Management</td>
</tr>
<tr>
<td>OV</td>
<td>OfficeVision/400</td>
</tr>
<tr>
<td>BRMS</td>
<td>Backup Recovery and Media Service/400</td>
</tr>
</tbody>
</table>

Other IBM software that may be used during part of the software life cycle includes:
Languages such as ILE RPG/400, ILE COBOL/400, and CL/400 can be used in a 3GL development environment.

V-RPG (Visual RPG) is another language, although not discussed in this redbook, that can be used to develop Client code on a PC with a Graphical User Interface (GUI). V-RPG fits in the source control mechanism of Application Development Manager as well, together with the host part of the application.

Programming Development Manager (PDM) is also an IBM software product that can be used in the design, coding, and maintenance stages of a software cycle.

Other software products, such as the entire SystemView family of products, is also very useful for production and maintenance support.

This document focuses on the use of specific IBM products and tools that help manage the software life cycle. Where no IBM tool exists for a particular process during the software life cycle (at the time of publication), suggestions are made for user-developed functions or tools. The following tools are described in more detail in this document:

- Application Development Manager
- Application Dictionary Services
- SM/400

### 1.4.3 Software Change Management

Software change management is concerned with the baseline and the control of changes to the parts that constitute a software product. A baseline is a collection of completed work items that are reviewed and agreed upon to serve as the basis for future development.

The most frustrating software problems are often caused by poor software configuration management. The problems are frustrating because they take time to fix, often happen at the worst time, and, in many cases, should have been avoided.

For example, a fully tested program suddenly does not work, or a difficult bug that was fixed at great expense suddenly reappears.

Application Development Manager offers an excellent solution in software configuration management. For more information, see Chapter 3, “Application Development Control Using ADM/400” on page 21.

By having a proper software configuration management tool such as Application Development Manager, unnecessary rework can be reduced to a minimum. This results in the following benefits when compared to a development project without a proper software configuration management tool:

- Reduced development cost
- Improved schedule
- Improved quality
1.4.4 Change Management and Impact Analysis

One of the most common problems in application development change management is the identification of programs and files that need to be changed. Application Dictionary Services is a good impact analysis tool that can help you design and maintain your applications in the following ways:

- To determine where fields are used in the objects.
- To determine what files, programs, or other objects are affected by a change to a given field.
- If you need to change a field to ensure that the change is reflected in all objects that refer to that field, and if necessary, to re-create files, programs, or other objects that are affected by this change.
- To search for strings in source members, or to scan for externally described field names in RPG programs to determine if a change to that file has real impact.
- To determine which programs are calling a specific program, or which programs are being called by a specific program.

By performing proper impact analysis in design, coding, testing, problem fixing, and maintenance, you can improve your productivity and quality of work.

1.4.5 Software Packaging, Distribution, and Maintenance

SM/400 allows you to package your application as a Licensed Program Product. The advantages to packaging your application with SM/400 is that you can distribute your application and fixes to your application in the same way that you distribute IBM Licensed Program Products and IBM PTFs. It can help you to simplify the installation of your application on your customers' systems, which can give your customer a more positive first impression of your product. You can track your PTFs the same way as you track IBM PTFs. You can add Licensed Management control into your applications so that you can charge your customers different prices for your product based on how people use the product or what AS/400 hardware it runs on. You can add symptom string and First Failure Data Capture information into your application to make PTF identification and enable automatic distribution of PTFs for reported problems that already have a PTF available.

1.4.6 Problem Management and Corrective Actions

The following are key elements in managing problems both before and after a software product is shipped:

1. Problems must be tracked with the following management-related information:
   - Problem severity level for allocating resource priority.
   - Problem status to indicate whether the problem is still opened, worked on, closed, or invalid.
   - The changed objects that are fixed for a problem.
   - Information on the name of the person that opened, assigned, and verified that a problem is resolved.
   - Additional information related to a problem at a different status. For example, comments on additional test requirements in subsequent test stages.
• The dates for the different status. This allows management actions for problems that are overdue and provide information on the service level in fixing problems.

2. The problem management tool should provide a summary status on all of the problems encountered in different stages of development. This provides information for a project manager to ensure that all of the problems are resolved before the release of a software product.

A problem management tool can also help to minimize problems with the next release of a software product by identifying “trouble areas” that need additional specification or testing.

OS/400 offers some ability to manage problems identified in the maintenance and support phase of a software life cycle. The problem management functions in SM/400 allow problem entry and subsequent update and tracking in a network of AS/400 systems. The Alert function of OS/400 also enables effective problem management at a central site for all of its defined remote sites.

1.4.7 Support Center/400

Support Center/400 (SC/400) is a problem/call management and incident tracking application for the AS/400 family.

SC/400 provides you the capability of tailoring your help desk solution to fulfill the current needs of your support structure. As your help desk grows, you can easily change or expand your existing implementation to address future requirements. Whether you are tracking problems, service requests, product enhancements, or usage assistance, SC/400 provides the key to a successful business environment.

1.4.7.1 SC/400 Features

SC/400 provides an effective and easy-to-use call management system, allowing for consistent incident reporting and automatic incident routing. With a solution database quickly accessed through a powerful search engine, response center agents who handle the initial call are enabled in many cases to resolve the end user’s problem. When a call must be transferred to a specialist, SC/400 automatically transfers the record to the appropriate individual or support group, thus eliminating the guesswork by the response center agent.

• Comprehensive Central Data Repository

SC/400 contains a central repository for various types of information. SC/400 maintains a database for all call records logged by the response center agents. As solutions to problems are created, SC/400 maintains a separate database for all solution records, thus greatly reducing the time spent rediscovering known solutions. The customer profile database contains all pertinent information regarding each customer serviced, including a note log that may contain helpful information that assists the agent in supporting the customer in a personalized manner.

Comprehensive inventory records and tracking data, which are part of the customer profile, can provide vital information to the response center agent at the time of the call. To assist response center agents who must work with multiple vendors (or applications) in their support environment, a vendor profile database may also be implemented. All of the preceding information is easily accessed while taking a call, increasing the efficiency of the agents who handle the initial calls.
• **Powerful Search Engine**

Provided with the SC/400 application is a powerful search engine that allows for expedient and accurate searches of database information. The response center agent can perform a search on any word that may have been recorded in the free-form text area of the call record.

• **Management Reports**

Included with SC/400 are standard reports that can be run at specific intervals for the effective management of your response center.

• **User Exits**

SC/400 provides the capability to access your own applications or programs interactively while recording or updating a call record.

• **Application Programming Interface**

Provided with SC/400 is an application programming interface (API) that allows the transferring of call record and solution record data to and from other systems.

In corrective action management, one of the major concerns in handling fixes for a problem is the ability to recover from a defective fix. The OS/400 PTF handling capability allows a PTF to be applied temporarily. If the PTF fix is tested and found to be defective, it can be removed easily by SM/400 without affecting the production system.

### 1.4.8 Project Management

Project Management is the planning, tracking, and controlling of a series of tasks to produce desired objectives. Normally, these objectives are measured in terms of:

- **Cost**: Plan versus actual for the total development cost.
- **Schedule**: Plan versus actual for the cycle time.
- **Quality**: Plan versus actual for defects after the general availability of a software product.

#### 1.4.8.1 Project Planning

A development plan defines the road map of activities that affect virtually every member of a project and is the cornerstone for communications across a project. Establishing a plan involves:

- Identifying and tailoring a development process model.
- Identifying all the tasks within a project.
- Establishing a project team.
- Estimating the effort and cost required for each task based on the experience and background of each team member.
- Setting the financial arrangements, contracts, machine, and real estate for the project.

Some of the preceding tasks cannot be automated. However, establishing a project plan can be automated by a project management tool such as the IBM Personal AS/2 Project Management (PAS/2). PAS/2 is a modular OS/2-based data analysis system. With the base module and the project management module, you can:
• Break down your project into tasks.
• Define dependencies and relationships between tasks.
• Define and assign resources for different tasks.
• Plan the project based on resource availability.
• Establish the cost of project based on the resource assigned.
• Make alternate plans and perform "what-if" analysis.
• Draw diagrams such as Gantt and Pert charts.
• Report progress and deviations in the project.

Figure 3. Example of a Gantt Chart Produced by Personal/AS Project Management

1.4.8.2 Project Tracking
Once the tasks are planned, the progress can be tracked using PAS/2. In tracking the progress, you can normally track the cost and schedule components. Tracking and predicting the software quality after General Availability requires the use of a mathematical model. See the IBM system journal article "Modeling and Software Development Quality," System Journal, Vol 30, Number 3, pages 351-362, for a statistical methodology in tracking and predicting software reliability quality.

1.4.8.3 Project Control
Project tracking and control are two very closely linked processes. An effective tracking process does not just discover problems; it also ensures that recovery plans are implemented and tracked before the problem starts to cause damage to the project. The goal of a recovery plan is to fix the problem quickly and limit the damage to the smallest area. It should address the following items:

• Identify the owner of a recovery plan.
• Identify the sequence of activities that must occur to complete the resolution.
• Determine the dates when each activity of the plan will be started and completed, and understand the dependencies of each activities.
• Ensure the appropriate people or groups approve and act on the plan.
1.4.9 Managing the Document Control Process

IBM OfficeVision/400 can be a useful tool during a software life cycle for the purpose of:

- Preparing and updating development documents such as requirements, functional specifications, program logic specifications, test plans, and test cases. These documents are stored online at the AS/400 system so that the latest versions are always available.
- Electronic mailing to facilitate communications between different project team members.
- Scheduling meetings and resources.

One of the most common problems in document control is the use of obsolete documents. For example, a programmer obtaining an obsolete design specification hard copy from the bottom of the filing cabinet ends up wasting a lot of effort. To avoid this problem, the following controls can be set up:

- Establish a register of all the online documentation indicating where the latest version of a document is located.
- Include document control elements such as the document issue number (edition notice), issue date, authors, and reviewers for a document. Refer to the front portion of this document for an example of document control elements.
- Arrange for the review and approval of the document.
- Issue the document and notify other project personnel that a document has been issued and any previous editions are obsolete.

Once a document control procedure is established and all team members are educated, common problems related to the document control process can be avoided. For example, a person reading a hard copy document can check the edition notice to ensure the latest document is being used. If there is any doubt, the register for all the documents can be searched to find the latest version and reprint.

1.4.10 Managing the Testing Process

Software testing often accounts for a major portion of an application development budget. Some of the greatest concerns in software testing are:

- Regression test
  A lot of test cases are completed for complex interactive programs to test the various combinations of scenarios. A program problem is reported and a fix developed. If the tester only performs the test to ensure the problem is fixed but does not perform regression test on other affected areas; one line of code change may result in multiple problems in other programs. The essence in managing testing is to ensure relevant test cases can be retrieved and rerun easily. A keystroke recording and replaying test tool such as the IBM Workstation Interactive Test Tools (WITT) can help in regression testing.
- Performance stress test
  Prerecorded WITT test cases can be run in performance stress testing, minimizing the need of operators required for the test.
- Test coverage
How do you know that all parts of your application are really tested and that all subroutines or program statements are executed at least once? Software Analysis Test Tool can help to:
- Generate test coverage queries and reports.
- Animate a program’s structure.
- Eliminate redundant or ineffective test cases.
- Interactive debug

CODE/400 allows interactive debug for programming languages such as COBOL/400, CL/400, RPG/400, and ILE languages. This enables a problem to be traced down to a program line level effectively.

1.4.11 Managing the Backup and Recovery Process
Software projects with a committed completion date must have an efficient backup and recovery plan to ensure fast recovery in case a disaster occurs. The IBM Backup Recovery and Media Service/400 (BRMS/400) can be used to ensure the development and maintenance environment can be quickly restored according to the policy defined by you. It provides AS/400 systems with support for policy-oriented setup and the implementation of archive, backup recovery, and other removable media-related operations.

Archive, backup, and recovery facilities enable users to establish how these operations are to be performed, with what level of controls, and with the flexibility for user extensions. It helps implement an effective and practical backup and archival strategy for performing general maintenance tasks on AS/400 systems, thereby minimizing the need for additional customer investment in media management. BRMS/400 provides cost savings, reduces backup complexity, and improves operational productivity.

1.4.12 Managing the Software Estimation Process
Software estimation is a very important component in any software life cycle. In most cases, project managers are asked to estimate the cost and duration of a project as soon as the requirement gathering and initial design are completed. These estimates are either required for external competitive bidding or used as cost estimates to build an initial business proposal for internal development.

If the estimated costs and duration are too high compared to others, the business will be lost. However, a winning bid with estimates that are too low causes a financial loss to the business and creates a bad image for the development organization. Accurate estimation is critical for all projects of this kind.


The key to software estimation is the collection of the following historical project information:
- Planned and actual effort for all of the tasks in a project. For example, 65 hours were spent in coding a particular Design Change Request compared to 60 hours planned.
• Planned and actual size of changes in design or coding. For example, 5,000 lines of code or the change is equivalent to 16 function points.

By collecting this information on effort and project size, a historical database can be set up to allow a better estimate to be performed for the next project.

1.5 Summary

• There are common application management problems:
  Examples of common application management problems and what solutions are offered.

• How you start changing your software life cycle:
  Do not start with huge projects, but take small pieces at a time, and recognize the areas where you have NO problems.

• What is the use of the example library delivered with this redbook:
  Do some hands-on training before you do your own pilot application.
Chapter 2. Mailing List Application Description

We use a modified version of the mailing list application as described in the Application Development by Example, SC41-9852, manual as the base application for the remainder of this document. The source code for the entire application is contained within the SG244187.1 library shipped with this Redbook.

Although not an actual customer application, the mailing list application used is defined in such a way that most readers should be able to associate its functions to what any application must do. This theoretical application creates and maintains a mailing list file (called the master file), prints mailing labels from the file, and provides an analysis of the file.

The original Mailing List application has been migrated to the ILE concept, but functionally still works the same as before. For more information about the migration process, see Moving to Integrated Language Environment for RPG IV, GG24-4358.

2.1 Functional Scenario

There are five main processes in the Mailing List application; these are:

1. Inquire into Mailing List Master file.
3. Submit mailing by account number.
4. Submit special analysis report.
5. Query Mailing List file.

To run the Mailing List application, call program MLGMNUC and the “Mailing List Menu” display is shown.

2.1.1 Inquire Into the Mailing List Master File

Option 1 on the “Mailing List Menu” shows you the “Mailing List Inquiry” display.
2.1.2 Maintain Mailing List Master File

Option 2 on the “Mailing List Menu” shows you the “Maintain Mailing List Master” display. From this display, you can:

- Display, change, add, or delete records in the Mailing List Master File.
- Display the Greater or Equal (GE) value.
- Do a name search.

For options 1 to 5, you have to enter an account number; for option 6 enter a name or part of a name in the Search Field.
Maintain Mailing List Master

<table>
<thead>
<tr>
<th>Action</th>
<th>Action</th>
<th>Account number</th>
<th>Account type</th>
<th>Name</th>
<th>Search name</th>
<th>Address</th>
<th>City</th>
<th>State</th>
<th>Zip code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1=Display</td>
<td>15902</td>
<td>2</td>
<td>Joseph Jones</td>
<td>JONES</td>
<td>3008 Brook St</td>
<td>Little Rock</td>
<td>AR</td>
<td>44877</td>
</tr>
<tr>
<td></td>
<td>2=Change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3=Add</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4=Delete</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5=Display GE value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6=Name search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For Options 1-5, enter an Account number
For Option 6, enter a Search field

F3=Exit

**Figure 7. Maintain Mailing List Master Display**

Following is an example display to “Change” the Mailing List Master file.

**Figure 8. Change Mailing List Master Display**

Action 5 on the “Maintain Mailing List Master” display allows you to search through the file (one record at a time) either from the account number entered or from the beginning of the file if no account number is entered. If no account number is entered, the search begins by displaying the first record in the file. You may page down to see the next record. If a value is entered, such as account number 50000, the search starts with account number 50000. If the specified account number does not exist, the first account number after 50000 is displayed. You can page up or page down. If you see a record you want to change or delete, function keys allow the display to be switched to change or delete mode.
Action 6 on the "Maintain Mailing List Master" display allows a method of determining the account number when only the account name is known. It allows you to search by entering one or more characters in the search field. For example, if you enter the name SMITH, all accounts with a search field of SMITH, SMITHERMAN, SMITHE, and so on are shown. This is based on the search field entered into every record.
You can respond by requesting a return with a specific account number. After the return, the record is displayed in “display mode.” You can press F6 to go into “change mode” on the record.

2.1.3 Submit Mailing by Account Number

Option 3 on the “Mailing List Menu” produces a listing of all of the records in the Mailing List Master file. See Figure 13 for an example.

```
<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Type</th>
<th>Name search</th>
</tr>
</thead>
<tbody>
<tr>
<td>15902</td>
<td>Joseph Jones</td>
<td>Little Rock</td>
<td>AR</td>
<td>44877</td>
<td>2</td>
<td>JONES</td>
</tr>
<tr>
<td>10057</td>
<td>Samuel Jones</td>
<td>Minneapolis</td>
<td>MN</td>
<td>55454</td>
<td>1</td>
<td>JONES</td>
</tr>
<tr>
<td>14477</td>
<td>Charles Hanley</td>
<td>Rochester</td>
<td>MN</td>
<td>55920</td>
<td>4</td>
<td>HANLEY</td>
</tr>
<tr>
<td>18090</td>
<td>Carol Larson</td>
<td>Rochester</td>
<td>MN</td>
<td>55920</td>
<td>5</td>
<td>LARSON</td>
</tr>
<tr>
<td>11458</td>
<td>James Grover</td>
<td>Trenton</td>
<td>NJ</td>
<td>08690</td>
<td>1</td>
<td>GROVER</td>
</tr>
<tr>
<td>26640</td>
<td>Daniel Benson</td>
<td>Syracuse</td>
<td>NY</td>
<td>13212</td>
<td>1</td>
<td>BENSON</td>
</tr>
<tr>
<td>38724</td>
<td>Maria Jonesa</td>
<td>Philadelphia</td>
<td>PA</td>
<td>22809</td>
<td>5</td>
<td>JONESA</td>
</tr>
<tr>
<td>24862</td>
<td>Kathryn Donty</td>
<td>Dallas</td>
<td>TX</td>
<td>75246</td>
<td>1</td>
<td>DONTY</td>
</tr>
</tbody>
</table>

Count of records- 9
```

Figure 13. Example Report

2.1.4 Submit Special Analysis Report

Option 4 on the “Mailing List Menu” produces a listing of all of the records in the Mailing List Master file with a zip code of 55920. See Figure 14 for an example.

```
<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Type</th>
<th>Name search</th>
</tr>
</thead>
<tbody>
<tr>
<td>14477</td>
<td>Charles Hanley</td>
<td>Rochester</td>
<td>MN</td>
<td>55920</td>
<td>4</td>
<td>HANLEY</td>
</tr>
<tr>
<td>18090</td>
<td>Carol Larson</td>
<td>Rochester</td>
<td>MN</td>
<td>55920</td>
<td>5</td>
<td>LARSON</td>
</tr>
</tbody>
</table>

Count of records- 2
```

Figure 14. Example Report Zip Code 55920
### 2.1.5 Query Mailing List File

Option 5 on the “Mailing List Menu” calls the STRQRY command and lets you create your own query against the Mailing List Master file.

### 2.2 Parts structure

A part is an object within Application Development Manager, such as an AS/400 file or program, or a source member containing RPG/400 code for an RPG program. Parts have system-supplied types, such as RPSRC, RPSINC, PGM, or FILE, or you can define your own types using an ADM/400 command. Parts can also have a system-supplied language, such as RPG, or SQLRPG, and you can define your own language for a user-defined part type using an ADM/400 command.

The following parts (objects) are part of the Mailing List application:

<table>
<thead>
<tr>
<th>Ship</th>
<th>ADM Part</th>
<th>ADM Part Type</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLGMNUC</td>
<td>CLLESRC</td>
<td>Mailing list menu program</td>
<td></td>
</tr>
<tr>
<td>MLGMTNC</td>
<td>CLLESRC</td>
<td>Mailing list maintenance</td>
<td></td>
</tr>
<tr>
<td>MLGRPTC</td>
<td>CLLESRC</td>
<td>Print one liner with MLGMSTL2 LF</td>
<td></td>
</tr>
<tr>
<td>MLGRPTC2</td>
<td>CLLESRC</td>
<td>General purpose query of MLGMSTP</td>
<td></td>
</tr>
<tr>
<td>MLGINQD</td>
<td>DDSSRC</td>
<td>Mailing list inquiry display</td>
<td></td>
</tr>
<tr>
<td>MLGMNUD</td>
<td>DDSSRC</td>
<td>Mailing list menu</td>
<td></td>
</tr>
<tr>
<td>MLGMSTL</td>
<td>DDSSRC</td>
<td>Mailing list label printing logical</td>
<td></td>
</tr>
<tr>
<td>MLGMSTL2</td>
<td>DDSSRC</td>
<td>Mailing list by state, city, search</td>
<td></td>
</tr>
<tr>
<td>MLGMSTL3</td>
<td>DDSSRC</td>
<td>General purpose LF for querying MLGMS</td>
<td></td>
</tr>
<tr>
<td>MLGMSTP</td>
<td>DDSSRC</td>
<td>Mailing master file</td>
<td></td>
</tr>
<tr>
<td>MLGMTND</td>
<td>DDSSRC</td>
<td>Mailing maintenance display file</td>
<td></td>
</tr>
<tr>
<td>MLGNAMD</td>
<td>DDSSRC</td>
<td>Mailing list name search display</td>
<td></td>
</tr>
<tr>
<td>MLGNAML</td>
<td>DDSSRC</td>
<td>Mailing list logical by name, state,</td>
<td></td>
</tr>
<tr>
<td>MLGREFP</td>
<td>DDSSRC</td>
<td>Mailing list field reference file</td>
<td></td>
</tr>
<tr>
<td>MLGINQR</td>
<td>RPGLESRC</td>
<td>Mailing list inquiry</td>
<td></td>
</tr>
<tr>
<td>MLGLBLR</td>
<td>RPGLESRC</td>
<td>Mailing list label printing ONE LABEL</td>
<td></td>
</tr>
<tr>
<td>MLGMTNR</td>
<td>RPGLESRC</td>
<td>Mailing list master maintenance</td>
<td></td>
</tr>
<tr>
<td>MLGNAMR</td>
<td>RPGLESRC</td>
<td>Mailing list name search</td>
<td></td>
</tr>
<tr>
<td>MLGRPTR</td>
<td>RPGLESRC</td>
<td>Mailing list one line report per name</td>
<td></td>
</tr>
<tr>
<td>MLGINQR</td>
<td>BLDOPT</td>
<td>Mailing inquiry</td>
<td></td>
</tr>
<tr>
<td>MLGMNUC</td>
<td>BLDOPT</td>
<td>Mailing menu</td>
<td></td>
</tr>
<tr>
<td>MLGMTNC</td>
<td>BLDOPT</td>
<td>Mailing maintenance</td>
<td></td>
</tr>
<tr>
<td>MLGRPTC</td>
<td>BLDOPT</td>
<td>Mailing report</td>
<td></td>
</tr>
<tr>
<td>MLGRPTC2</td>
<td>BLDOPT</td>
<td>Mailing report - 2</td>
<td></td>
</tr>
<tr>
<td>PGMDFTBLD</td>
<td>BLDOPT</td>
<td>Default Build option for programs</td>
<td></td>
</tr>
<tr>
<td>QDFT</td>
<td>BLDOPT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 15 (Part 1 of 2). Parts Used in the Base of the Mailing List Application*
| Y | MLGINQR  | PGM | Mailing list inquiry                  |
| Y | MLGMNUC  | PGM | Mailing list menu program             |
| Y | MLGMTNC  | PGM | Mailing list maintenance              |
| Y | MLGRPTC  | PGM | Print one liner with MLGMSTL2 LF      |
| Y | MLGRPTC2 | PGM | General purpose query of MLGMSTP      |

---

MLGINQR MODULE Mailing list inquiry
MLGLBLR MODULE Mailing list label printing ONE LABEL
MLGMNUC MODULE Mailing list menu program
MLGMTNC MODULE Mailing list maintenance
MLGMTNR MODULE Mailing list master maintenance
MLGNAMR MODULE Mailing list name search
MLGRPTC MODULE Print one liner with MLGMSTL2 LF
MLGRPTC2 MODULE General purpose query of MLGMSTP
MLGRPTR MODULE Mailing list one line report per name

---

Y MLGMSGF MSGF Mailing list message file for MLS support

---

Y MLGINQD FILE Mailing list inquiry display
Y MLGMNUD FILE Mailing list menu
Y MLGMSTL FILE Mailing list label printing logical
Y MLGMSTL2 FILE Mailing list by state, city, search
Y MLGMSTL3 FILE General purpose LF for querying MLGMS
Y MLGMSTP FILE Mailing master file
Y MLGMTND FILE Mailing maintenance display file
Y MLGNAMD FILE Mailing list name search display
Y MLGNAML FILE Mailing list logical by name, state,
Y MLGREFP FILE Mailing list field reference file

---

CREATE PARTL Partlist for the Automatic Initial Build
CRTOUT PARTL PARTL output from the Initial Create

---

Ship = Y, means that this is eligible for shipping as part of the executable application.

Figure 15 (Part 2 of 2). Parts Used in the Base of the Mailing List Application
Chapter 3. Application Development Control Using ADM/400

Today, change is intrinsic to most software engineering endeavors. Applications consist of many components, some of which are based on previous versions and all of which may be undergoing constant revision. The problems of application development are further complicated by the interdependencies that exist between components. Changing just one component may result in changes to several other components that depend on it.

To build and maintain high-quality applications efficiently, application development organizations must have a consistent and systematic approach to manage changes they are making to their applications. They are looking for standard methods and procedures and, where possible, automation to improve productivity and reduce backlogs.

Development organizations need to be able to organize and manage all of the components of an application as a unit. They also need to be able to control the baseline or master version of an application. They often want to control multiple versions of components or of entire applications. And they want to be able to make quick fixes to the code they are developing. They need a mechanism that allows for shared access to components, and they want the means to plan and manage their entire development process. They must reduce the time they spend in maintaining applications so that they have more time to develop new applications.

3.1 Application Development Manager

The Application Development Manager product answers many of the needs of today’s application developers. It provides a team of application developers working in an AS/400 environment with a mechanism for efficiently and effectively managing their development environment and its application objects throughout the life of the application.

Benefits

- **A standard development process**
  A development team can define the application that suits its organization and methods.

- **Increased productivity**
  This product organizes both the developers writing the code and the code to be written. As developers write their code and compile and test it, they work efficiently and productively in a well-organized development environment where changes to their code are managed.

- **Flexibility and versatility**
  The structure defined at the beginning of a project does not restrict the development team. This structure is changed and refined at any time as the demands of the project change. Developers are added to or removed from the project, and code is shared and reused.

- **Support for several versions of an application**
  Developers can create and maintain multiple versions of an application in both the Application Development Manager development environment and in
a production environment. They can easily identify which versions of source
and objects belong to a particular version of the application.

- **An automated build process**
  Developers can rely on the powerful build process to build, or compile, the
  source code for an application more quickly. They no longer have to analyze
  the relationships between pieces of code; the build process does this for
  them automatically.

- **Data security and integrity**
  The structure that the Application Development Manager environment
  provides ensures the integrity and security of production, test, and
  development versions of the code. Developers are able to work with the
  different versions of the code. And they work in an environment where they
  are assured that they cannot overwrite one another’s changes.

- **An audit trail**
  A project log records what has changed in the application, the commands
  used to change the hierarchy or components it contains, who issued the
  commands, and when the activity took place.

- **Notification about the status of any component of the application**
  Developers receive messages that tell them that a part they are requesting
  is already being changed, that it exists in another branch of the project
  hierarchy, and that a change made in one version may also have to be made
  in the other version. This is useful when fixes to a production version of a
  part have to be propagated to the follow-up version of the part. The person
  doing this work needs to know where the part is in the hierarchy and who
  has checked it out to a development group.

  An other nice command to help with your fixes is the “Merge Part”
  (MRG PART) command. This is ADM/400’s interface to the File Compare and
  Merge Utility. This command merges specific parts or all of the parts in the
  default search path from a specified target group, and can be particularly
  useful if you must migrate a change from a fix version of a part to a follow-on
  version of a part, or to merge vendor changes with your own changes for a
  given part.

- **Archiving**
  Developers can choose to use the archive function when they promote a
  source to a higher level. If a version of that source exists already at that
  level, that version is archived before it is replaced.

- **Ability to package applications**
  The Application Development Manager product provides a mechanism that
  assists in the packaging process. The various solutions for packaging and
  distribution are discussed in Chapter 4, “Distributing and Supporting Your
  Application Using ADM/400” on page 31, Chapter 6, “Package the NLS
  Release of Your Application Using Application Development Manager and
  SM/400” on page 49, and Chapter 7, “Packaging Without the Use of
  ADM/400” on page 71.
3.1.1.1 Who is Using Application Development Manager/400

The two main types of users for this product are:

- **The project administrator**
  
  The person who creates a project is automatically authorized to work on the project as the project administrator. This person defines the phases, such as development and testing, through which components of an application go before they are actually placed in a production environment. This person performs the following actions:
  
  - Maintain the project hierarchy, divided into groups.
  - Enroll and remove developers from the project and grant them authority to the different groups.

- **The developer**
  
  The application developer is any member of the development team who has been given access to the application by the project administrator. Application developers usually only have **update** access to specific groups. They do the following tasks:
  
  - Create or change source code (parts) of the application in a development group. Parts to be modified are copied from the application group already under control of Application Development Manager.
  - Compile parts and applications using the build process that automates much of this work.
  - Test the application (or parts), either within the control of Application Development Manager or outside by moving the code to another test environment.

See 13.1, “Tips Using Application Development Manager for the ILE Application Development” on page 169 for more information and considerations in using Application Development Manager and the ILE concept.

We have chosen the Application Development Manager product as the base for our development environment. The Application Development Manager Introduction and Planning Guide, GC09-1807, gives more detailed information on the usage of this product.

3.2 The Mailing List Application and ADM/400

This topic covers the setup of our environment and the import of this application into Application Development Manager. The guidelines for using Application Development Manager and the setup considerations are well documented in the Application Development Manager/400 User and the Application Development Manager Introduction and Planning Guide. The topics that are discussed are:

- The base project hierarchy.
- Import the existing mailing application into Application Development Manager.
- Build the application.
- Test the application.

The complete import and building of the application in Application Development Manager is done by:

```sql
CALL SG244187.1/CRTADMMENV
```
or you can do all of the manual steps as they are explained in the rest of this chapter. If you use the create program, test the contents of the CRTOUT part list (Figure 20) and the Build report (Figure 19), and finally run the application.

### 3.2.1 The Base Project Hierarchy

![Diagram](image)

Figure 16. The Hierarchy of the Base Project

We created the Project MLA and the other groups as shown in Figure 16, with the following commands:

- `CRTPRJ PRJ(MLA) SHORTPRJ(MLA) SAVDTA(*YES) TEXT('Mailing List Application')`
- `CRTGRP PRJ(MLA) GRP(PRODUCTION) SHORTGRP(PRD) PARENT(*NONE) PRMCODE(V1) NOTIFY(*DEVELOPER) TEXT('Production group for project MLA')`
- `CRTGRP PRJ(MLA) GRP(TEST) SHORTGRP(TST) PARENT(PRODUCTION) PRMCODE(V1) NOTIFY(*DEVELOPER) TEXT('Test group for project MLA')`
- `CRTGRP PRJ(MLA) GRP(DEVELOPMENT) SHORTGRP(TST) PARENT(TEST) PRMCODE(V1) NOTIFY(*DEVELOPER) TEXT('Development group for project MLA')`

### 3.2.2 Importing the Application
Figure 17. IMPPART Command to Import Parts

The import is done directly into the production group assuming that the application is a well-running application today. If for some reason you need to maintain the application, you can still use the TEST and DEVELOPMENT group. If the promotion to the production level is done using the Archive function, the original imported part is retained.

In our example, we cannot use the import function for *ALL objects from our library since we want to import parts that have no equivalent in the non-ADM environment such as, for instance, a Buildoption (BLDOPT). They are actually stored as a CLP in a sourcefile, and if incorrectly imported, become a CL part.

Note:

You can get non-ADM equivalent parts to be imported if you store them before import in specific-named source files such as QBLDOPTSRC.

The following commands are used to import the application:

/* Import the sources into the Production Group */

IMPPART OBJ(SG244187.1/QMLGSRC) OBJTYPE(*ALL) MBR(*ALL)
     PRJ(MLA) GRP(PRODUCTION) SRCFILE(*TYPE) +
     DATA(*YES) TEXT(*TEXT)

/* Import the Messagefile into the Production Group */

IMPPART OBJ(SG244187.1/MLGMSGF) OBJTYPE(*MSGF) +
     PRJ(MLA) GRP(PRODUCTION) SRCFILE(*TYPE) +
     DATA(*YES) TEXT(*TEXT)

/* Import the BLDOPT sources into the Production Group */

IMPPART OBJ(SG244187.1/QMLGBLDOPT) OBJTYPE(*SRC) MBR(*ALL)
     PRJ(MLA) GRP(PRODUCTION) TYPE(BLDOPT) +
     LANG(*ATTR) SRCFILE(*TYPE) DATA(*YES) +
     TEXT(*TEXT)
TEXT('TEXT') is specified so that the text description of each imported object or database member is preserved after the import process. Unique part text is often used to easily identify the part when packaging or modifying the application.

If TEXT('Importing Mailing List application') is specified, all imported parts have the same text field contents - "Importing Mailing List application!"

### 3.2.3 Building the Application

![Build Part (BLDPART)](image)

After the application is successfully imported into group PRODUCTION, the application must be built in Application Development Manager. This causes all application parts to be compiled. The build process creates the relationships between source and object, as well as between components. We used the following command to rebuild the complete application in Application Development Manager:

```
BLDPART PRJ(MLA) GRP(PRODUCTION) TYPE(PARTL) PART(CREATE) +
SAVLST(*YES) PARTL(CRTOUT)
```

*Figure 18. Rebuilding the Complete Application*
A PARTL is a part that contains a list of other parts. In general, the commands that allow you to work with a PARTL, also work with the content of the PARTL such as:

- **BLDPART (Build Part)**
  At the PART keyword, the part list content is used to control which parts are used in the build process.
  At the PARTL keyword, you specify that all of the parts built by this BLDPART action are added to a part list.

- **PRMPART (Promote Part)**
  At the PART keyword, the part list content is used to control which parts are promoted.
  At the PARTLOPT, you specify whether you want to only use the content (*LIST) of the PARTL, the PARTL part itself (*PART), or *BOTH.
  At the PARTL keyword, you specify that all of the parts promoted are added to a part list.

- **EXPPART (Export Part)**
  At the PART keyword, the part list content is used to control which parts are exported.
  At the PARTL keyword, you specify that all of the parts promoted are added to a part list.

- For OTHER commands related to parts, see the Application Development Manager/400 User manual.

This part list can be used for checking the build results. It can also serve as a base list of the parts that need to be distributed to some other systems. See Section 4.3.3, “Edit the CRTOUT Part List in Group PRODUCTION” on page 36 for more information.

In our case, the building process completed successfully and we received the message “24 parts built, 0 parts failed, 0 warnings.” To see the details of the build process, look in the QPLYBLDP spooled file. An example of such a report is shown in Figure 19.
Figure 19. QPLYBLDP Output After Building

The part list that was created as a result of the BLDPART looks the same as:
This build process has created all of the necessary objects for our application; we created modules and the programs were created from the modules according to our supplied build options. So finally our application structure looks the same as the one shown in Figure 21.

Figure 21. Initial Program Structure

Now the application is ready to be tested. Use option 45, Add project library list on the “Work with Parts Using PDM” display to set the library list, or use the following command:

ADDPRJLIBL PRJ(MLA) GRP(PRODUCTION)
Add Project Library List (ADDPRJLIBL)
Type choices, press Enter.
Project ................................ > MLA Name
Group ................................. > PRODUCTION Name
Scan hierarchy ...................... > *YES *YES, *NO
Search path ......................... > *DFT *DFT, name

F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys

Figure 22. Add Project Library List Command

Use option 16 (Run) against the MLGMNUC program part as shown in Figure 23 to verify and test if the rebuild of the application in group Production was successful.

Work with Parts Using PDM
Project .......... MLA
Specified group ... PRODUCTION
Position to ...... Position to type ........

Type options, press Enter.
2=Change 3=Copy 4=Delete 5=Display 6=Print 7=Rename
8=Display information 13=Change information 14=Build 16=Run ...

Opt Part Type Language Group
USLOBJCMD BLDOPT *NONE PRODUCTION
MLGINQR PGM RPGLE PRODUCTION
16 MLGMNUC PGM CLLE PRODUCTION
MLGMTNC PGM CLLE PRODUCTION
MLGRPTC PGM CLLE PRODUCTION
MLGRPTC2 PGM CLLE PRODUCTION
MLGINQR MODULE RPGLE PRODUCTION

Parameters or command
====>
F3=Exit F4=Prompt F5=Refresh F6=Create
F9=Retrieve F10=Command entry F23=More options F24=More keys

Figure 23. Run the Main Menu Program

3.2.4 Summary

The use of the Application Development Manager product:

• Project hierarchy
• Importing an existing application
• Using part lists for control
In OS/400 Version 3 Release 7, Version 3 Release 6, and Version 3 Release 2, there are two methods available to distribute and support your application.

The first method is to use a new function provided in Application Development Manager that allows you to distribute your application objects through SNADS to multiple AS/400 systems. This chapter describes how Application Development Manager can be used to accomplish the distribution of your application objects.

The second method, which has been available for several releases, is to use SM/400 to “package” your application for distribution and support. The SM/400 processes used to “package” your application are discussed in Chapter 6, “Package the NLS Release of Your Application Using Application Development Manager and SM/400” on page 49.

In general, you should consider using the SM/400 method of packaging your application since it is a much more robust solution. Some of the things you should consider before choosing which distribution method is best for you are:

- The number of objects in your application.
- The stability of your application, that is, how often you expect to have to fix problems or to refresh the entire application with a new release.
- The number of systems on which you have to support your application.
- Whether or not your application includes folders.

You should keep these criteria in mind and review this chapter as well as Chapter 6, “Package the NLS Release of Your Application Using Application Development Manager and SM/400” on page 49 and Chapter 8, “Servicing Your Products With the AS/400 PTF Process” on page 87 before choosing the distribution method that is most suitable for your situation.

Note

In order to distribute your application or fixes using the Application Development Manager support, your development system must be at OS/400 release V3R7, V3R6, or V3R2.

You do not need an Application Development Manager license on each system to which you intend to distribute code. You can copy the required commands and code to each target system, including V3R1 systems, that enables Application Development Manager to provide electronic distribution for your application or fixes.
4.1 Advantages and Disadvantages

This section lists some of the advantages and disadvantages of using Application Development Manager for distributing and supporting your code.

4.1.1 Advantages

• Application Development Manager can provide basic electronic distribution of applications or fixes without the need to install any other IBM or vendor products.

• The process used to distribute applications or fixes is quite simple; it is done using only two commands (EXPPART and RCVPART).

• The process can be easily modified to add or remove systems by maintaining just one Application Development Manager part (of type SYSTEML).

• You can maintain multiple system lists or have system lists in multiple project hierarchies or search paths within Application Development Manager.

4.1.2 Disadvantages

• Since the Application Development Manager solution is quite simple, you may have to develop additional programs to handle such functions as archival of changed objects, changes to database files, and so on.

• There is less traceability for your fixes than in a combined Application Development Manager and SM/400 solution, which is described in Chapter 6, “Package the NLS Release of Your Application Using Application Development Manager and SM/400” on page 49 and Chapter 8, “Servicing Your Products With the AS/400 PTF Process” on page 87.

4.2 Prerequisites

Before you can start distributing your application objects, you need to satisfy some prerequisites:

• Network setup

• Install the RCVPART code on all target systems.

4.2.1 Network Setup

When you use the EXPPART command to distribute to a list of systems, Application Development Manager uses SNADS functions as the distribution mechanism. Therefore, you must configure SNADS connections between your development system and each system to which you intend to distribute your application.

4.2.2 Install RCVPART Code on All Target Systems

In order to receive distributions using the Receive Part (RCVPART) command from remote AS/400 systems, a subset of the Application Development Manager programs that support this command must first be installed on the remote systems. The remote AS/400 systems must also be at V3R7, V3R6, V3R2 or V3R1.
If the remote AS/400 systems are at V3R7, V3R6 or V3R2 and have the Application Development Manager feature installed, skip this section.

4.2.2.1 Installing RCVPART Code on V3R7, V3R6, V3R2 System Without Application Development Manager

To distribute the receive programs that support the Receive Part (RCVPART) command to a a V3R7, V3R6, or V3R2 AS/400 system that does not have the Application Development Manager feature installed, you must do the following:

1. At the sending system (a system with the V3R7, V3R6, or V3R2 Application Development Manager feature already installed), create a *SAVF QADMDIST in the QADM library by typing:

   ```plaintext
   CALL QADM/QLYSAVDST
   ```

   **Note:** In order to use this command successfully, you need to have *SECADM and *ALLOBJ authority.

   The QLYSAVDST program then completes the following tasks:
   - Creates the QADMDIST library.
   - Copies a set of objects from the QADM into the QADMDIST library.
   - Creates a *SAVF called QADMDIST in the QADM library.
   - Saves the QADMDIST library into the *SAVF using the *CURRENT value on the Target release parameter of the Save Library (SAVLIB) command.
   - Deletes the QADMDIST library.

2. Send the QADMDIST *SAVF to each remote V3R7, V3R6, or V3R2 system where you want to use the Receive Part (RCVPART) command:

   ```plaintext
   SNDNETF FILE(QADM/QADMDIST) TOUSRID((AABBCC SYSNAM2))
   ```

3. At the receiving system, receive the *SAVF:
   - Create a *SAVF:
     ```plaintext
     CRTSAVF FILE(QGPL/QADMDIST)
     ```
   - Receive the network file:
     ```plaintext
     RCVNETF FROMFILE(QADMDIST) TOFILE(QGPL/QADMDIST)
     ```

4. On the receiving system, restore the *SAVF into the QADMDIST library, using the Restore Library (RSTLIB) command:

   ```plaintext
   RSTLIB SAVLIB(QADMDIST) DEV(*SAVF) SAVF(QGPL/QADMDIST)
   ```

4.2.2.2 Installing RCVPART Code on a V3R1 System With or Without Application Development Manager

To distribute the needed programs that support the Receive Part (RCVPART) command to a V3R1 AS/400 system with or without the Application Development Manager feature installed, you must do the following:

1. At the sending system (at the V3R7, V3R6, or V3R2 with the Application Development Manager feature installed), create a save file QADMDISTP in the QADM library by typing:

   ```plaintext
   CALL QADM/QLYSAVDSTP
   ```

   **Note:** In order to use this command successfully, you need to have *SECADM and *ALLOBJ authority. The QLYSAVDSTP program then completes the following tasks:
   - Creates the QADMDIST library.
   - Copies a set of objects from the QADM into the QADMDIST library.
   - Creates a *SAVF called QADMDIST in the QADM library.
   - Saves the QADMDIST library into the *SAVF using the *CURRENT value on the Target release parameter of the Save Library (SAVLIB) command.
   - Deletes the QADMDIST library.

2. Send the QADMDIST *SAVF to each remote V3R7, V3R6, or V3R2 system where you want to use the Receive Part (RCVPART) command:

   ```plaintext
   SNDNETF FILE(QADM/QADMDISTP) TOUSRID((AABBCC SYSNAM2))
   ```

3. At the receiving system, receive the *SAVF:
   - Create a *SAVF:
     ```plaintext
     CRTSAVF FILE(QGPL/QADMDISTP)
     ```
   - Receive the network file:
     ```plaintext
     RCVNETF FROMFILE(QADMDISTP) TOFILE(QGPL/QADMDISTP)
     ```

4. On the receiving system, restore the *SAVF into the QADMDIST library, using the Restore Library (RSTLIB) command:

   ```plaintext
   RSTLIB SAVLIB(QADMDISTP) DEV(*SAVF) SAVF(QGPL/QADMDISTP)
   ```
• Creates a *SAVF called QADMDISTP in the QADM library.

• Saves a list of objects from the QADMDISTP library into the *SAVF using the *PRV value on the Target release parameter of the Save Object (SAVOBJ) command.

2. Send the QADMDISTP *SAVF from the QADM library to each remote V3R1 system from which you want to use the Receive Part (RCVPART) command:
   
   SNDNETF FILE(QADM/QADMDISTP) TOUSRID((AABBCC SYSNAM1))

3. Send the QADMPGM *SAVF from the QADMDISTP library to each remote V3R1 system from which you want to use the Receive Part (RCVPART) command:
   
   SNDNETF FILE(QADMDISTP/QADMPGM) TOUSRID((AABBCC SYSNAM1))

4. At the receiving system, receive these *SAVFs:

   • Create the *SAVFs:
     
     CRTSAVF FILE(QGPL/QADMDISTP)
     CRTSAVF FILE(QGPL/QADMPGM)

   • Receive the network file:
     
     RCVNETF FROMFILE(QADMDISTP) TOFILE(QGPL/QADMDISTP)
     RCVNETF FROMFILE(QADMPGM) TOFILE(QGPL/QADMPGM)

5. On the receiving system, create the QADMDISTP library:
   
   CRTLIB QADMDISTP

6. Restore the *SAVF QADMDISTP into the QADMDISTP library using the Restore Object (RSTOBJ) command:
   
   RSTOBJ OBJ(*ALL) SAVLIB(QADMDISTP) DEV(*SAVF) SAVF(QGPL/QADMDISTP)

7. Restore the *SAVF QADMPGM into the QADMDISTP library using the Restore Object (RSTOBJ) command:
   
   RSTOBJ OBJ(*ALL) SAVLIB(QADMDISTP) DEV(*SAVF) SAVF(QGPL/QADMPGM)

---

4.3 Exporting Your Application to a List of Systems

In the following sections, we go through the steps to export the Mailing List Application to a library called MLALIB on a remote system.

4.3.1 The SYSTEML Part Type

Application Development Manager now supports a part type of SYSTEML. This part type is used with the EXPPART command to distribute parts to one or more systems that are specified within the SYSTEML part. For example, there is one system list part for each release of the application being stored in the project. There is one system list part for the current release containing the list of systems on the current release, and another one for the previous release, and so on.

You can use the CRTPART command to create a SYSTEML part:

CRTPART PRJ(MLA) GROUP(PRODUCTION) TYPE(SYSTEML) PART(SYSTEMS) TEXT(‘system list for EXPPART’)

Now you can edit the SYSTEML part by taking option 2 against the part from the Work with Parts Using PDM display. The edit option takes you into an SEU
session. You can type in the list of systems to which you want to distribute, one system per line, as shown in Figure 24 on page 35.

```
Columns . . . :  1  80                        Edit

SEU==>                     FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...
*************** Beginning of data ****************************
0001.00 SYSNAMA
0002.00 SYSNAMB
0003.00 SYSNAMC
0004.00 SYSNAMD
0005.00 SYSNAME
*************** End of data ****************************
```

Figure 24. Example SYSTEML Part

4.3.2 Create an Installation Program for Your Application (Optional)

If your application requires any additional configuration after the objects are installed on the target system (or systems), you should develop the installation program on the development system and distribute it with the application objects.

Things to consider for your installation program are:

- Create the data library and copy the empty files from the application library into the data library. If the application already exists on the target system (or systems), any existing data should be retained.

- Create any additional objects (data queues, and so on) that are required by your application, but are not directly supported by Application Development Manager.

- Change ownership of specific objects in your application. You can specify a default owner for all objects on the EXPPART command; this owner is attached to all of the objects in your application. If specific objects need an owner that is different than the default you specify on the EXPPART command, you can change the owner in your installation program.

- Change authority of specific objects in your application. You can specify a default authority for all of the objects on the EXPPART command; this authority is *PUBLIC for all objects in your application. If *PUBLIC needs a different authority than the default you specify on the EXPPART command, you can change it with your installation program.

- You should not put commands in your installation program that run only in an interactive environment.
4.3.3 Edit the CRTOUT Part List in Group PRODUCTION

Change part list CRTOUT so that all parts to be exported are included. If you do not want to export the MODULE objects, you can remove them from the part list. You can use the user-defined options XS (to select the CRTOUT PARTL), XA (to add parts), and XR (to remove parts) on the Work with Parts Using PDM display if you have installed the ADDPART program as defined in Appendix B, “Utility programs” on page 257. If you have not installed the ADDPART program, you can edit the part list part by typing a 2 next to the part list name. You are taken into a DFU session where you can add entries into and remove entries from the part list. To delete an entry using DFU, you must be in CHANGE mode, find the entry, and then press F23 twice. To add entries, you must be in ENTRY mode.

In our case, you need to add part MLGMSGF, type MSGF, and an installation program (if you created one) to the part list CRTOUT and then continue with the export.

4.3.4 Exporting the Application Using the System List

You can now use the EXPPART command to send your application objects to the list of systems.

```
Type choices, press Enter.
Project . . . . . . . . . . . . > MLA
Group . . . . . . . . . . . . . > PRODUCTION
Type . . . . . . . . . . . . . . > PARTL
Part . . . . . . . . . . . . . . > CRTOUT
Language . . . . . . *ALL
Change date . . . . *BEGIN
Scan hierarchy . . . . *YES
Search path . . . . . . . . . . *DFT
Cross project . . . . . . . . . . *NO
To library . . . . . . . . . . . > MLALIB
To source file . . . . . . . . . > FROMFILE
Copy data . . . . . . . . . . . *NO
Replace object . . . . . . . . . . *YES
Owner of exported part . . . . > QSECOFR
Authority . . . . . . . . . . *CHANGE
PARTL processing option . . . > *LIST
System list . . . . . . . . . . > SYSTEMS
Target release . . . . . . . . . . *CURRENT

F3=Exit F4=Prompt F5=Refresh F12=Cancel F13=How to use this display
F24=More keys
```

Figure 25. Exporting to a SYSTEML

The objects listed in part list CRTOUT are packaged into a save file called QADMSAVF and the save file is sent to the same user ID on all of the remote systems listed in the SYSTEML part as the user ID on the central system who is running the EXPPART.

Requirements:

- The user ID on the development system where you invoke the EXPPART must also exist on all of the systems in the SYSTEML. This is the user ID to which the QADMSAVF *SAVF is sent.
• The library specified on the TOLIB parameter on the EXPPART command must exist on each target system before the RCVPART command is run.

• The user ID on the target system to which the QADMSAVF is sent must have the correct level of authority to the TOLIB library and to the OWNER user profile on each target system.

• EXPPART with the use of a SYSTEML and RCVPART both require library QTEMP to be empty.

4.3.5 Run the RCVPART Command on Each Target System

Once the EXPPART command has been run using the SYSTEML on the development system, you can sign on to each target system and run the RCVPART command.

On the V3R7, V3R6 or V3R2 target system:

• In order to receive an Application Development Manager distribution using RCVPART, you must be signed on as the user who sent it. In our example, we use QSECOFR. This is required in this example because the OWNER parameter on the EXPPART command was specified as QSECOFR.

• Create the library for the application:
  CRTLIB MLALIB

• Add the QADMDIST library to your library list; this is only necessary if Application Development Manager is not installed:
  ADDLIBE QADMDIST

• Invoke the RCVPART command:
  RCVPART

The parts are restored from save file QADMSAVF into the library that was specified on the EXPPART command.

You can also set up the RCVPART command as a batch job; this gives you the flexibility to run the command, for example, during the night.

4.3.6 RCVPART Support for V3R1

• The best approach for previous release support is a separate group within the project for building with the previous release, and a separate SYSTEML part for the previous release target systems.

• If the target system for the EXPPART command is a V3R1 system, the objects should all be compiled with a target release of V3R1M0. The TGTRLS parameter on the EXPPART command also needs to be specified as V3R1M0.

• You should add the QADMDISTP library to your library list:
  ADDLIBE QADMDISTP

4.3.7 Run Your Install Program (Optional)

If you have an install program that was sent along with the application, you can run it at this point.
4.4 Support Your Code After Install

The technique to distribute fixes is the same as the technique previously outlined to distribute your entire application. The steps are:

- Create a maintenance hierarchy in ADM/400 in which to make your changes.
- Check out the part (or parts) to be changed.
- Make the changes, compile and test the changes, and promote them back to the appropriate group.
- Create a part list that includes all of the changed parts.
- Run EXPPART against the part list, specifying the appropriate SYSTEML part.
- Sign on to each target system to which you sent the changed objects and run the RCVPART command to install the changes.

**Note**

Even if the REPLACE (replace object) parameter on the EXPPORT command is set to *YES, physical files are not replaced. RCVPART recognizes this and does not attempt to replace files. The file is sent as a NETFILE to the user-ID who invoked the RCVPART so the file can be manually received and replaced.

---

4.5 Maintaining an Audit Trail of Changes

You may want to consider putting some kind of a naming or numbering convention in place to track your code fixes. For instance, if you are using a problem tracking tool such as SC/400, you can use the SC/400 problem record number in the part list name for the corresponding fix. Using a method such as this allows you to track your fixes by relating the objects in the part list back to the problem that they were changed to fix.

---

4.6 Summary

- SYSTEML part type contains a list of systems to which you intend to export code.
- Specify the SYSTEML on the EXPPART command to distribute to target systems.
- Use RCVPART on target systems to receive code.
Chapter 5. Enhance the Application for National Language Support

The National Language Support Planning Guide explains in detail how to write applications that are National Language Support enabled or multilingual. Here we just summarize the various techniques for coding textual data.

5.1.1 Direct Coding

The most common way to define constant text is to specify the text directly in the source code as a literal. This technique can be used for:

- **Display files**
  - Constants such as titles, instruction lines, option definitions, headings, field prompts, and command key descriptions.
  - Default values on input fields (DFT keyword).
  - Error messages (ERRMSG/SFLMSG keyword).
- **Printer files**
  - Constants such as titles, headings, and total line descriptions.
- **User commands**
  - Prompt descriptions on the command definition statements.

The source members need to be translated and the objects created for different languages. The programs that are not language dependent are in one library and the language-dependent objects are in a separate library. When the application is run, you can choose the appropriate textual data of the language version you want to work with by setting up the library list with the specific library containing the textual data and the program library.

5.1.2 Early Binding of Messages

Text can be stored externally from the source code in a separate message file, but is bound into the object when it is created. This technique can be used for:

- **Display files**
  - Constants such as titles, instruction lines, option definitions, headings, field prompts, and command key descriptions.
- **Printer files**
  - Constants such as titles, headings, and total line descriptions.
- **User commands**
  - Prompt descriptions on the command definition statements.

For display and printer files, the message is referred to by the MSGCON keyword in the DDS source specifications.

For user commands, the message identifier is specified on the PROMPT keyword instead of a literal, and the message file is referred to on the Create (CRTCMD) command.

This technique allows you to create any number of objects in different languages using the same source code by just assigning another message file at object creation time. The message file is only needed during the creation of the object.
5.1.3 Late Binding of Messages

Text can be stored externally from the DDS source code in a message and bound only to the display format at runtime. This technique can be used for:

- **Display files only**
  - Constants such as titles, instruction lines, option definitions, headings, field prompts, and command key descriptions (MSGID keyword).
  - Default values on input fields (MSGID keyword).
  - Field validation specifications (CHKMSGID keyword).
  - Error messages (ERRMSGID/SFLMSGID keyword).

In the DDS for the display file, the message is specified through the MSGID keyword.

With this technique, you create any number of message files in different languages and different libraries with only one DDS source code and display file object. During runtime, you assign another message file by setting the library list accordingly.

5.1.4 Storing Text in Database Files

Text can be stored externally from the source code in a database file, retrieved by the application program, and moved to the display or print format at runtime. This technique can be used for:

- **Display files**
  - All constant text.
  - Default values on input fields.
  - Error messages.

- **Printer files**
  - All constant text.

- **Programs**
  - All constants such as compare values, scan characters, and tables.

With this technique, you create any number of database files in different languages and different libraries with only one DDS source code and display file object. During runtime, you assign the corresponding database file by setting the library list accordingly.

5.2 Our National Language Support Project Hierarchy

In the base mailing application, we use two techniques to code the textual data in the English version.

1. 5.1.1, "Direct Coding" for the RPG programs MLGMTNR and MLGRPTR.
2. 5.1.3, "Late Binding of Messages" for the error messages and prompts in the Display Files.

Coding literals in RPG programs that are displayed on a panel is, of course, not a wise thing to do if you want to enable your application for National Language Support. But the Mailing List application is an existing application, and we think this reflects real life. There are still many applications such as this. For ease of use, we have only one message file with the messages used by the MSGID and ERRMSGID keywords. With the National Language Support scenario, we test the different language packaging scenarios using SM/400.
If we add a National Language Support version to our application, the base project setup in Application Development Manager is not adequate anymore. We do not want to mix up those different language environments so we add two new branches with three groups each to our Project “MLA.”

We add one for the English NLS version and one for Dutch. The use of the three branches is:

1. PRODUCTION, which contains all of the Non-NLS code.
2. PRODUCTION_ENGLISH, which contains the English NLS code.
3. PRODUCTION_DUTCH, which contains the Dutch NLS code.

![Diagram of NLS Hierarchy of the Project]

Application Development Manager does not allow you to make changes to a product structure while someone is working in that product. So, if you are on an Application Development Manager display, you need to press F3 repeatedly until you have backed out of Application Development Manager. Then, you can execute commands that change the product’s structure.

```plaintext
CRTGRP PRJ(MLA) GRP(PRODUCTION_ENGLISH) SHORTGRP(PRDE) PARENT(PRODUCTION)
PRMCODE(ENGLISH) NOTIFY(*DEVELOPER) PARTLREQ(*NO)
TEXT(‘English NLS version’)

CRTGRP PRJ(MLA) GRP(TEST_ENGLISH) SHORTGRP(TSTE) PARENT(PRODUCTION_ENGLISH)
PRMCODE(ENGLISH) NOTIFY(*DEVELOPER) PARTLREQ(*NO)
TEXT(‘English NLS version’)

CRTGRP PRJ(MLA) GRP(DEVELOPMENT_ENGLISH) SHORTGRP(DEVE) PARENT(TEST_ENGLISH)
```
The first new groups have the promote code “ENGLISH,” so the parts checked out to one of these groups can never be promoted beyond group PRODUCTION_ENGLISH. The same is true for the construction of the DUTCH groups.

You can also call a program called:

CALL SG244187.1/CRTADMNL5

5.3 Transition Into the NLS Environment

For the transition to the NLS solution, we have to do a couple of other things besides just checking out parts. We need to re-assign the parts depending on their dependency to the non-NLS or NLS environment.

The sequence to follow is:

1. Check out the NLS parts to an NLS environment (DUTCH).
2. Delete all of the NLS parts from the Non-NLS environment.
3. Copy all of the parts from one NLS environment (DUTCH) to the other NLS environment (ENGLISH).
4. Change the parts in the NLS Dutch environment.
5. Build the parts in the NLS Dutch and English environment.
6. Promote the parts to the appropriate level in the hierarchy.

5.3.1 Checkout NLS Parts

Use option 12 to select the "DEVELOPMENT_DUTCH" group from the group list in project MLA.
Figure 27. Check Out Parts to Group DEVELOPMENT_DUTCH

To check out the parts, use option 28 for every part as shown in Figure 27 and use Roll to get to the other parts as well.

Check out message file:
- MLGMSGF

and the two RPG programs that use literals to be translated:
- MLGMTNR
- MLGRPTR

**Tip**

After the checkout, you see on the parts located in the different groups on the “Work with Parts in PDM” display. It is very convenient to work only with the parts in the DEVELOPMENT_DUTCH group. You can take two different actions to get the display as shown in Figure 29 on page 45, for example:

1. Use F18 to change the Scan hierarchy from “Y” to “N”.
2. Use F17 and define “DEVELOPMENT_DUTCH” at the group level.

### 5.3.2 Delete the NLS Parts for the Non-NLS Environment

Since you do not need the NLS related parts in the Non-NLS environment anymore, you should delete them now, otherwise you might find that a build done later in the Non-NLS environment completes with the wrong parts built. For instance, if you still leave the modules of an NLS-related part, the build of a program that needs that module succeeds when actually it should fail because it should be built in the NLS environment only.
The following checklist gives you some guidance on how to determine the parts that should be deleted. Using the Work with Parts Using PDM display for project MLG, group PRODUCTION:

1. Delete the parts that you just checked out:
   - MLGMSGF
   - MLGMTNR
   - MLGRPTR
2. Delete the related parts built such as:
   - Module MLGMTNR
   - Module MLGRPTR
3. Delete the parts that relate to the module parts:
   - Program MLGMTNC
   - Program MLGRPTC
   - Program MLGRPTC2

Another way to determine what needs to be deleted is to use the BLDPART command for TYPE(*ALL) and PART(*ALL) in the NLS Dutch environment and detect all of the parts that are built. They should be deleted from the Non-NLS environment.

**Note:** You cannot copy from one NLS environment to another before you have deleted the parts from a higher level.

### 5.3.3 Copy the NLS Parts for the English Environment

Use the copy option “3” from the DEVELOPMENT_DUTCH group and fill in the next display.

![Copy Parts](image)

Now the transition to NLS has been completed.
5.3.4 Change the NLS Parts

Work with Parts Using PDM

<table>
<thead>
<tr>
<th>Project</th>
<th>MLA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specified group</td>
<td>DEVELOPMENT_DUTCH</td>
</tr>
<tr>
<td>Position to</td>
<td>Position to type</td>
</tr>
</tbody>
</table>

Type options, press Enter.

2=Change   3=Copy   4=Delete   5=Display   6=Print   7=Rename
8=Display information   13=Change information   14=Build   16=Run ...

<table>
<thead>
<tr>
<th>Opt</th>
<th>Part</th>
<th>Type</th>
<th>Language</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MLGMTNR</td>
<td>RPGLESRC</td>
<td>RPGLE</td>
<td>DEVELOPMENT_DUTCH</td>
</tr>
<tr>
<td>1</td>
<td>MLGRPTR</td>
<td>RPGLESRC</td>
<td>RPGLE</td>
<td>DEVELOPMENT_DUTCH</td>
</tr>
<tr>
<td>2</td>
<td>MLGMSGF</td>
<td>MSGF</td>
<td>*NONE</td>
<td>DEVELOPMENT_DUTCH</td>
</tr>
</tbody>
</table>

Parameters or command

F3=Exit   F4=Prompt   F5=Refresh   F6=Create
F9=Retrieve   F10=Command entry   F23=More options   F24=More keys

Figure 29. Result of the Checkout Process

With the message file in the group DEVELOPMENT_DUTCH, we can start to translate the message descriptions. Use option 2, Change, on the “Work with Parts Using PDM” display to change the message file. When we finish the translation of the message file, use option 2 again for both the RPG sources and translate the literals in the sources. The parts are now ready to be built.

Tip

In the library SG244187.1, you find a message file called MLGMSGD; this is a translated Dutch version. Copy this message file over the MLGMSGF message file in library MLG.DEVD using the CRTDUPOBJ command. Then, make sure to go back into ADM and use option 2 to change the message file, to make it “stale” within ADM. In the source file QSRCCH5 in the same library, are the changed RPG source members for this exercise.
5.3.5 Build and Test the NLS Parts

Build Part (BLDPART)
Type choices, press Enter.
Project ............... > MLA Name
Group ................... > DEVELOPMENT_DUTCH
Type ................... > *ALL Name, *generic*, *ALL...
Part .................... > *ALL Name, *generic*, *ALL
Search path ............. > *DFT *DFT, name
Scope of build .......... > *NORMAL *NORMAL, *LIMITED, *EXTENDED
Force build .............. *NO *NO, *YES
Build mode .............. *COND *COND, *UNCOND, *RPTONLY
Save list ............... *YES *NO, *YES
Perform bind step ....... *YES *YES, *NO

Figure 30. BLDPART Command for the Translated Parts

This example shows how Application Development Manager can detect the changed parts since the last build and only build the parts that have changed by specifying TYPE(*ALL) and PART(*ALL). Application Development Manager builds all of the parts changed for National Language Support automatically.

You could have maintained your own change list and built each part separately, such as specifying TYPE(RPGLESRC) and PART(MLGNTNR). But why not let Application Development Manager do it for you! Remember that spooled file QPLYBLDP lists the results of the build process.

After all of the parts are built in this group, we can test the Dutch version of the application. First use option 45, Add project library list, on the “Work with Parts Using PDM” display to set the library list correctly. See Figure 22 on page 30 for an example. Test the Dutch version of the application with option 16 (Run).

Work with Parts Using PDM
Project .............. MLA
Specified group ....... DEVELOPMENT_DUTCH
Position to ............ Position to type ......
Type options, press Enter.
2=Change 4=Delete 5=Display 14=Build
16=Run 28=Check out 29=Check in 30=Promote ...

Opt Part Type Language Group
MLGINQR PGM RPGLE PRODUCTION
MLGLBLR PGM RPGLE PRODUCTION
16 MLGMNUC PGM CLLE PRODUCTION
MLGMTNC PGM CLLE PRODUCTION
MLGMTNR PGM RPGLE DEVELOPMENT_DUTCH
MLGNAMR PGM RPGLE PRODUCTION
MLGRPTC PGM CLLE PRODUCTION

Parameters or command
===>
F3=Exit F4=Prompt F5=Refresh F6=Create
F9=Retrieve F10=Command entry F23=More options F24=More keys

Figure 31. Run the Dutch Version of the Application
5.3.6 Promote Parts

After testing, the parts in group DEVELOPMENT_DUTCH can be promoted to the next group in the hierarchy with the command:

```
PRMPART PRJ(MLA) GRP(DEVELOPMENT_DUTCH) TYPE(*ALL) PART(*ALL)
    EXTEND(*YES)
```

You should use F5 = Refresh on the Work with Parts using PDM display after doing the PRMPART command to see that the parts have been promoted.

With the EXTEND(*YES) parameter, not only the source parts are promoted but also their associated output parts. Parts that are generated by building source parts are output parts and can only be promoted if you use the EXTEND parameter on the Promote Parts command. EXTEND(*YES) is the only means by which parts with a promote code of *NONE can be promoted. One exception to the rule is: Parts that were not built directly from source such as a program from a module, are not promoted with EXTEND (*YES), but need to be re-built at a hierarchy level.

Repeat the same command to get the parts promoted to the PRODUCTION_DUTCH group:

```
PRMPART PRJ(MLA) GRP(TEST_DUTCH) TYPE(*ALL) PART(*ALL) EXTEND(*YES)
```

You should use F5 = Refresh on the Work with Parts using PDM display after doing the PRMPART command to see that the parts have been promoted.

You can also promote only the source parts to the next group in the hierarchy, do a build and test in that group, promote the source again to the production group, and do the final build.

In our example, one test at the DEVELOPMENT group level is enough, so we use the parameter EXTEND (*YES) to promote the source and object parts together. When the parts have arrived at the PRODUCTION_DUTCH level, run the following command:

```
BLDPART PRJ(MLA) GRP(PRODUCTION_DUTCH) TYPE(*ALL) PART(*ALL)
```

Now three parts, namely the programs, are rebuilt and finally you have to delete the NOT promoted programs for the DEVELOPMENT_DUTCH group.

5.4 How To Do the Second NLS (English) Version

To create the second NLS (in this case, English), you need to repeat the following sections.

- 5.3.4, “Change the NLS Parts”
- 5.3.5, “Build and Test the NLS Parts”
- 5.3.6, “Promote Parts”

For English, the exception is that the original parts are already in that language, so skip 5.3.4, “Change the NLS Parts.”
5.5 Summary

- National Language Support design should be done early in the development process because it affects the way in which you code your application.

- You should create Application Development Manager hierarchies for each language that you intend to support. Any NLS parts, such as message files, should be copied in each NLS hierarchy. This does imply that you have to maintain multiple copies of the same part.
This chapter demonstrates how to use functions provided in Application Development Manager to invoke the SM/400 commands that result in a “packaged” product. By packaging your application as a product, you enable it for support through the same OS/400 commands and processes that are used to support IBM products. Some of the advantages of packaging your applications in this way are:

- There is no need for you or your customers to understand an application support scheme that is unique to your application. If you or your customers know how to load IBM products and PTFs, they understand how to load and update your application.
- Once your product is packaged, you can quickly tell what the latest fix level is for any system on which it is installed. This can reduce the amount of time required to diagnose problems for which fixes already exist.
- Distribution of your application and fixes to your application can be done either electronically or on tape, and can be controlled from a central system.
- Through special exit programs that you provide as part of your application and fixes to your application, you can simplify and automate many of the processes for installing and maintaining your application.
- You can break your application up into different pieces (options), and have a separate pricing structure for each piece.
- You can roll back defective fixes easily.

Before proceeding with the next sections of this book, we recommend that you read chapter 2, Product Packaging - Initial Steps, chapter 3, Packaging Complex Products, chapter 5, Product Installation, Distribution, and Management, and chapter 10, Controlling Operations Centrally in the System Manager/400 Use to learn the concepts of product packaging.

### 6.1 Mailing List Application Packaging Library Structure

Figure 32 on page 50 shows the library and folder structure of the Mailing List application on the packaging system. These are the libraries where the objects are exported to by Application Development Manager. To the product packaging process, these are the development libraries and folders. Note that the packaging development libraries and folders are not the same as the Application Development Manager development libraries. The objects and documents ready to be packaged reside in the packaging development libraries and folders.
AS/400 Packaging Development Environment

<table>
<thead>
<tr>
<th>Non-language sensitive objects</th>
<th>English language objects</th>
<th>Dutch language objects</th>
<th>Empty library</th>
<th>Source files for both:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGM - LF - PF - DSPF</td>
<td>MSGF - PGM</td>
<td>MSGF - PGM</td>
<td>MLGDATA LIB</td>
<td>MLGSRCLIB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MLGCODELIB</th>
<th>MLG2924</th>
<th>MLG2923</th>
<th>MLGDATA LIB</th>
<th>MLGSRCLIB</th>
</tr>
</thead>
</table>

Folder with English documents
MLGFLR/MLG2924F
Folder with Dutch documents
MLGFLR/MLG2923F

**Figure 32. Mailing List Application Packaging Libraries and Folders**

### Library/Folder Name Contents

**MLGCODELIB**

This contains all of the RPG and CL programs that do not have any language-sensitive Machine Readable Information (MRI). Physical and logical files in this library are part of the product, but do not contain the customer data once the product is installed. We refer to these products’ files as “file shells”. The application objects in this library are:

- MLGREFP - PF
- MLGMSTP - PF
- MLGMSTL - LF
- MLGMSTL2 - LF
- MLGMSTL3 - LF
- MLGNAML - LF
- MLGINQD - DSPF
- MLGMTND - DSPF
- MLGMNUD - DSPF
- MLGNAMD - DSPF
- MLGMNUC - PGM(CLLE)
- MLGINQRP - PGM(CLLE)

In addition, the following objects that are described in this chapter are added to this library.

- MLGPRODDFN - PRDDFN
- MLGLODCODB - PRDLOD
- RSTLIC110 - PGM(CL)
- USCRTUSRPG - PGM(RPG)
• USLOBJRPG - PGM(RPG)
• USRTVUSRPG - PGM(RPG)
• SZRTVLIB1 - PGM(CL)
• USCRTUSCMD - CMD
• USLOBJCMD - CMD
• SZRTVLIB - CMD

MLGDATALIB
This contains the database files with customer data after the product is installed, but this library is empty when the product is packaged. During installation, a postoperation exit program is run to duplicate all of the physical and logical files from the MLGCODELIB library to this library. The MLGDATALIB is planned to be used as the production data library for the Mailing List application, but the database files in this library are created after the product is installed and are not part of the product.

This design facilitates the following operations after the Mailing List application is installed:
• Release upgrade
  When a new release of your product is installed to replace a previous release, all of the product’s objects are replaced whether they have changed or not. If you replace large database files whose formats have not changed, the installation process can become longer and more cumbersome; installation exit programs must save and restore the data even if the files have not changed. By not making the files where the user data is stored part of the product, you can control the migration of data files to the new release through exit programs, affecting only those files that have changed.
• By including the empty data library as part of the product, you can enable dynamic naming for this library. When the product is installed, a different name for this library can be specified, but OS/400 still tracks this library as part of the product. Installation and PTF exit programs can retrieve the name of the installed library.
• Some characteristics of the PTF process justify this design.

MLGSRCLIB
This contains the source files for the product that can be optionally shipped to or be installed on the remote site systems.

Note
Normally, you do not want to ship source code with your application. We are doing this only to be able to give an example of how you package an option for your product.

MLG2924 This contains the language-sensitive objects (MRI) for English (message files, display files, and programs with English constants). The objects in this library are:
• MLGMSGF - MSGF
• MLGMTNC - PGM(CLE)
• MLGRPTC - PGM(CLE)
• MLGRPTC2 - PGM(CLE)
• MLGPKGMSGF - MSGF
This contains the language-sensitive objects (MRI) for Dutch (message files, display files, and programs with Dutch constants). The objects in this library are:

- MLGMSGF - MSGF
- MLGMTNC - PGM(CLLE)
- MLGRPTC - PGM(CLLE)
- MLGRPTC2 - PGM(CLLE)
- MLGPKGMSGF - MSGF
- MLGLODLNGD - PRDLOD

Documents with English MRI are stored in the MLGFLR/MLG2924F folder.

Documents with Dutch MRI are stored in the MLGFLR/MLG2924F folder.

### 6.2 Converting Your Application Into a Packaged Product

In the following sections, we discuss how to convert your application into a product.

#### 6.2.1 Packaging Process Overview

On the AS/400 system, an application consists of a set of objects in one or more libraries and, optionally, of documents in folders. The documents can be office documents, help text, or PC files. When packaged, the application becomes a product, and we refer to the packaged application as a product to distinguish it from an application that has not been packaged. Packaged applications are enabled for the same level of software management services as IBM Licensed Program Products, so you can distribute, install, and maintain your product using the IBM process. Enhancements to the process that IBM develops for IBM Licensed Program Products are made available to customers through enhancements to OS/400, SM/400, and MSS/400. Therefore, a central site system with SM/400 installed (service provider) can deliver service to remote systems (service requesters) in a fashion similar to IBM Service Support. Neither MSS/400 or SM/400 is required on the service requester systems if you only need to provide the same level of support for your product as for IBM Service Support.

In this section, we provide an overview of the process for packaging, distributing, and maintaining an application. Following sections describe specific tasks we used in our scenario for the Mailing List application. For a more detailed description of product structure, packaging considerations, and the packaging product process, refer to System Manager/400 Use.

Whether you use OS/400 APIs, SM/400 commands, or Application Development Manager export PARTL parts to package your applications, the objects that must be created to package your product and the internal process that takes place are the same.

A **product** consists of one or more options. When the product is installed, at least the `*BASE` option must be installed. The additional options are installed only if the functions that they provide are needed by the user.
An Option is a logical group of library objects and folders. The objects and folders that make up the product option belong to one or more product loads. Every product option has a Code Load that contains the object code for that option. It can also have additional language loads (one per National Language supported) if the application has been enabled for translation. A language load (or loads) contains the language-sensitive objects (Machine Readable Information, (MRI)) such as display files, message files, printer files, and so on.

To package an application into a product, the following steps must be followed:

1. Create the product message file. This message file contains the message descriptions that are displayed when you look at your software inventory using the Display Software Resources (DSPSFWRSC) command. There must be one message file for each national language supported and one message description for each product option.

2. Create a product definition object, object type *PRDDFN (one for each product). When using Application Development Manager to create your product, you create a part of type PRDDFN in your Application Development Manager hierarchy, and the *PRDDFN object is created automatically when you export your application.

3. Create the exit programs that are used to automate the tasks associated with saving, installing, and deleting your product.

4. Create the Product Load objects, object type *PRDLOD. There must be one product load LODTYPE(*CODE) for each product option. Language loads are optional; there must be one language load, LODTYPE(*LNG), for each product option for each national language supported by the product if it is national language enabled. When using Application Development Manager to create your product, you create a part of type PRDLOD in your Application Development Manager hierarchy, and the *PRDLOD is created automatically when you export your application.

5. The objects that belong to each product load must be marked by changing the object description, also known as object information repository (OIR), using the Change Product Object Description (CHGPRDOBJD) command. This command adds information to the object description identifying the product ID, the Version/Release/Modification level, the option, and the load ID the object belongs to. The product definition (*PRDDFN) and product load (*PRDLOD) objects must also be marked. When you use Application Development Manager to create your product, the CHGPRDOBJD command is run automatically for each object that is exported, including the *PRDDFN and *PRDLOD objects.

6. Package each product option using the Package Product Option (PKGPRDOPT) command. When you use Application Development Manager to package your product, the PKGPRDOPT command is run automatically for each PRDLOD part that is exported.

7. Save your product to tape or *SAVF (SAVLI CPGM).

8. Install your product (RSTLICPGM).

System Manager/400 Use provides a detailed description of the product packaging process. Refer to Figure 33 on page 54 for an overview picture of a product’s structure and product packaging process.
Figure 33. Product Structure and Product Packaging Process Overview

- **PRODUCT**
  - **BASE**
  - **OPTION**

- **PRODUCT BASE OPTION PRIMARY LIBRARY**
  - OPT LOAD
  - OBJTYPE: PRODUCT DEFINITION
  - OBJTYPE: MRM PRODUCT LOAD
  - OBJTYPE: MRI PRODUCT LOAD
  - OBJTYPE: MRTFI
  - OBJTYPE: PRODUCT OBJECTS

- **OPTION ADDITIONAL LIBRARIES**
  - 1 2 3 10

- **OPTION ROOT FOLDER, SUBFOLDERS, AND DOCUMENTS**
  - 1 2 ...

- **OPTION PRIMARY LIBRARY**
  - LOAD
  - OBJTYPE: MRM
  - OBJTYPE: MRI

- **OPTION ADDITIONAL LIBRARIES**
  - 1 2 ...

- **SECONDARY OPTION PRIMARY LIBRARY**
  - LOAD
  - OBJTYPE: MRM
  - OBJTYPE: MRI

- **SECONDARY LIBRARIES FOR OPTION CONTAINS**
  - PRODUCT OBJECTS

- **FOLDERS FOR OPTION CONTAINS**
  - DOCUMENTS
  - SUBFOLDERS WHICH CONTAIN DOCUMENTS

WHSE MRM = NONE LANGUAGE-SENSITIVE LOAD
MRI = LANGUAGE-SENSITIVE LOAD/S

NC138 04/92
6.3 Product Packaging Using ADM/400

Application Development Manager provides two part types to enable you to package your application using SM/400. The part types are called product definition (PRDDFN) and product load (PRDLOD). The product definition contains general information about the product; the product load contains information about a specific product option load. See 6.2, “Converting Your Application Into a Packaged Product” on page 52 for detailed information.

For packaging to take place during an export operation, a part type PRDLOD must be found in a part of type PARTL. If a product load part is found, the parts listed in the PARTL part are packaged. The product ID is specified on the Product ID parameter of the Create Product Load (CRTPRDLOD) command inside the part of type PRDLOD.

The steps to export and package the application are:

1. Create a product definition and product load parts.
2. Create the message files needed for packaging.
3. Create the exit program (or programs) as required.
4. Edit the PARTL parts, add the names of the PRDDFN, PRDLOD, message files, and exit program (or programs).
5. Export the PARTL part and package the application.

6.3.1 Create a Product Definition and Product Load Part

To create a product definition part, use the following commands:

1. CRTPART command.

   CRTPART PRJ(MLA) GRP(PRODUCTION) TYPE(PRDDFN) PART(MLGPRODDFN) TEXT(‘product definition MLG application’)

2. Use the CHGPART command or option 2 on the “Work with Parts Using PDM” display to add the CRTPRDDFN command and all of its parameters and values to the PRDDFN part.

   Tip

   The source code for the parts changed in this section can be copied from file SG244187.1/QSRCCH5, where the member name in the source file is the same as the name of the part being changed.
Note

Note that we allowed dynamic naming for our product. Dynamic naming says that the product libraries can be specified dynamically at the time that RSTLICPGM is run. That is, the end user can decide in which libraries the product is to be installed. If you allow dynamic naming for your product, you should be sure that you do not have any library qualified calls within your product.

To create a product load part, use the following commands:

3. CRTPART command:

   CRTPART PRJ(MLA) GRP(PRODUCTION) TYPE(PRLDOD) PART(MLGLODCODB) TEXT(*base product load, lodtype *code MLG application)

4. Use the CHGPUART command or option 2 on the “Work with Parts Using PDM” display to add the CRTPRDLOD command and all of its parameters and values to the PRDLOD part.

Figure 34. CRTPRDLDFN Command in the PRDDFN Part

Figure 35. CRTPRDLDO Command in the PRDLOD Part
Repeat steps 3 and 4 for every additional PRDLOD you have to create. In our example, we need four additional PRDLOD type parts:

1. Part name MLGLODLNGE in group PRODUCTION_ENGLISH:

   CRTPART PRJ(MLA) GRP(PRODUCTION_ENGLISH) TYPE(PRDLOD) PART(MLGLODLNGE)
   TEXT(English lng load, lodtype *LNG MLG application)

   Now edit the part and add:
   CRTPRDLOD PRDLOD(MLGLODLNGB) PRDID(1MLG001) RLS(V1R1M0) OPTION(*BASE)
   LODTYPE(*LNG) LODID(2924) RGSID(*PHONE 123456789) DVLLIB(MLG2924)
   PRILIB(*CODE) MINTGTRLS(V3R1M0) LNGLIB(MLG2924)
   FLRL(('MLGFLR/MLG2924F'))

2. Part name MLGLODLNGD in group PRODUCTION_DUTCH:

   CRTPART PRJ(MLA) GRP(PRODUCTION_DUTCH) TYPE(PRDLOD) PART(MLGLODLNGD)
   TEXT(Dutch lng load, lodtype *LNG MLG application)

   Now edit the part and add:
   CRTPRDLOD PRDLOD(MLGLODLNGB) PRDID(1MLG001) RLS(V1R1M0) OPTION(*BASE)
   LODTYPE(*LNG) LODID(2923) RGSID(*PHONE 123456789) DVLLIB(MLG2923)
   PRILIB(*CODE) MINTGTRLS(V3R1M0) LNGLIB(MLG2923)
   FLRL(('MLGFLR/MLG2923F'))

3. Part name MLGLOD1E in group PRODUCTION_ENGLISH:

   CRTPART PRJ(MLA) GRP(PRODUCTION_ENGLISH) TYPE(PRDLOD) PART(MLGLOD1E)
   TEXT(Option 1, English source, MLG application)

   Now edit the part and add:
   CRTPRDLOD PRDLOD(MLGLOD1) PRDID(1MLG001) RLS(V1R1M0) OPTION(1)
   LODTYPE(*CODE) LODID(5002) RGSID(*PHONE 123456789) DVLLIB(MLGSRCLIB)
   MINTGTRLS(V3R1M0)

4. Part name MLGLOD1D in group PRODUCTION_DUTCH:

   CRTPART PRJ(MLA) GRP(PRODUCTION_DUTCH) TYPE(PRDLOD) PART(MLGLOD1D)
   TEXT(Option 1, Dutch source, MLG application)

   Now edit the part and add:
   CRTPRDLOD PRDLOD(MLGLOD1) PRDID(1MLG001) RLS(V1R1M0) OPTION(1)
   LODTYPE(*CODE) LODID(5002) RGSID(*PHONE 123456789) DVLLIB(MLGSRCLIB)
   MINTGTRLS(V3R1M0)

**Note**

Specify the MINTGTRLS for the earliest release of OS/400 on which you intend to support your product.

Specify the real principal development library name in the DVLLIB parameter; do not use *PRDDFN or *CODE. In some occasions, Application Development Manager does not create the *PRDLOD in the proper library if you use *PRDDFN or *CODE.

Specify different names for the PRDLOD Parts. Application Development Manager does not allow you to have parts with the same name and type in the project hierarchy.
6.3.2 Create the Message File

We need two message files. They contain the message descriptions that are displayed when you use the DSPSFWRSC command after the product is installed with the RSTLICPGM command.

The command to create the first PART, type MSGF is:

```
CRTPART PRJ(MLA) GRP(PRODUCTION_ENGLISH) TYPE(MSGF) PART(MLGPKGMSGF) TEXT('packaging message file')
```

For packaging, the name of the message files (MSGF) in the English and Dutch Language load have to be the same. Copy the MSGF MLGPKGMSGF from group PRODUCTION_ENGLISH to PRODUCTION_DUTCH.

After the creation, use option 2 on the “Work with Parts Using PDM” display to change the message files in both groups and add the message descriptions to the files. In MSGF MLGPKGMSGF in group PRODUCTION_ENGLISH, create message descriptions:

- MSG0001 - MSG('Mailing application *BASE')
- MSG0002 - MSG('Mailing application Option 1')

In MSGF MLGPKGMSGF in group PRODUCTION_DUTCH, create message descriptions:

- MSG0001 - MSG('Verzendlijst applicatie Optie *BASE')
- MSG0002 - MSG('Verzendlijst applicatie Optie 1')

6.3.3 Create the Postoperation Exit Program

The postoperation exit program (RSTLIC110) and all of the commands and programs it calls are created in another project hierarchy and imported to the PRODUCTION group.

For this example, you should import the following parts into the group PRODUCTION from library SG224187.1:

- RSTLIC110 - PGM
- USCRTUSRPG - PGM
- USLOBJRPG - PGM
- USRTVUSRPG - PGM
- SZRTVLIB1 - PGM
- USCRTUSCMD -CMD
- USLOBJCMD -CMD
- SZRTVLIB -CMD

Use the following program to do the import:

```
CALL SG244187.1/IMP_EXIT6
```
6.3.3.1 Sample Exit Program RSTLIC110

We use a postoperation Exit program that runs after the product is restored using the RSTLICPGM command. The program RSTLIC110 (CLP) is defined as a postoperation program (POSTOPRPGM parameter in the CRTPRDLOD command) and its main purpose is to copy the "file shells" from the base option’s primary library (MLGCODELIB) to the base option’s first additional library. That is where the user’s data files reside after the product is installed (MLGDATALIB). By copying the files after the product is installed into this library, the files in the production data library are not part of the product.

Notice that since we allow dynamic naming in our product definition (ALWDYNNAM parameter in the CRTPRDDFN command), we must retrieve the names of the restored libraries in our exit program; we cannot assume that the names are the same as the ones specified in the product loads.

The following summarizes the steps performed by the exit program RSTLIC110:

1. List all of the physical and logical files in the primary library (MLGCODELIB or installed library name if the name was overridden during the RSTLICPGM operation).

2. Duplicate the files into the product’s data library (MLGDATALIB or installed additional library name if the name was overridden during the RSTLICPGM operation).

3. Add product information to the object descriptions (OIR); this is the information about each object that identifies which product it is associated with, such as release level, option, and load identifier. This is done for documentation purposes only, since the files duplicated into MLGDATALIB are not part of the product.

The exit program RSTLIC110 calls other commands and programs to perform these functions. The exit program and all of the commands and programs it calls are part of the product and we package them with the product in our example.

Another alternative for creating the files in the data production library is to do the preparation work at the central site where you package your product and save the files into a Save File in MLGCODELIB. This save file should be part of the product and should ship with it. The postoperation exit program should only restore the files from the SAVF into MLGDATALIB instead of creating the files and updating the object description.

Note

You can refer to Chapter 5 in System Manager/400 Use for a detailed description of exit program functions and the error messages that are sent from exit programs. To keep our sample exit program simple, we are not including the handling of error situations that should be considered in a real production environment.
/* PGM Name RSTLIC110 */
/* PGM Function: Post-operation EXIT program for RSTLICPGM */
/* This is an example of an exit program for the */
/* 1st installation of the product. */
/* Scenario: Only database file descriptions ("file shell") */
/* are part of the product and shipped in the product’s primary */
/* library MLGCODELIB. */
/* After installing the LPP this EXIT program duplicates product’s */
/* data base files (LF and PF) in the Product’s data library. */
/* This way the data base files where the user data is stored are */
/* not part of the product */

/*1) Create the user space that will contain the list of objects */
/* (Data Base Files) to be duplicated. */
/* API : QUSCRTUS */
/* RPG PGM : USCRTUSRPG */
/* CMD : USCRTUSCMD */

/*2) List all the Objects type *FILE in the product library */
/* in the User Space */
/* API : QUSLOBJ */
/* RPG PGM : USLOBJRPG */
/* CMD : USLOBJCMD */

/*3) Retrieve the Object List stored in the User Space */
/* API : QUSRVTUS */
/* RPG PGM : USRVUSRPG */

/*4) Get the Installed Library Name for MLGDATALIB to duplicate */
/* the files (SZRTVLIB) and pass control over to USTRVUSRPG */
/* CL PGM : SZRTVLIB (CPP) */
/* CMD : SZRTVLIB */
/* API : QSZRTVPR */

/*5) Duplicate the Objects Type *File Special Attribute PF or LF */
/* CRTDUPOBJ */

/*6) Update the object information (for documentation purpose only) */
/* to contain product ID and LOAD ID */
/* API : QLICOBJD */
/* RPG PGM : USQLICOBJ */

PGM PARM(&FUNCTION &INDICATOR &LNGID &EXITPGMLIB +
CURRENTLIB &CREATELIB &RSTLIB &CURRENTFLR+
CREATEFLR &RSTFLR &CURVRM &RSTVRM+
LMISSOBJ &NRMISSOBJ &LMISSFLR &NRMISSFLR)

Figure 36 (Part 1 of 2). Sample Exit Program RSTLIC110
6.3.4 Create and Edit the PARTL Parts

We created five PARTL parts and used these PARTL parts to do the packaging. Perform the following steps:

- The MLGOBJ PARTL contains the objects that need no National Language Support translation:
  
  ```plaintext
  CRTPART PRJ(MLA) GRP(PRODUCTION) TYPE(PARTL) PART(MLGOBJ)
  TEXT('PARTL for all the NONMRI objects')
  ```

  The MLG2924 PARTL contains language-sensitive objects for the English language load:
  
  ```plaintext
  CRTPART PRJ(MLA) GRP(PRODUCTION_ENGLISH) TYPE(PARTL) PART(MLG2924)
  TEXT('PARTL for all the English MRI objects')
  ```

  The SRC2924 PARTL contains all of the English source code (RPG, DDS, and so on):
  
  ```plaintext
  CRTPART PRJ(MLA) GRP(PRODUCTION_ENGLISH) TYPE(PARTL) PART(SRC2924)
  TEXT('PARTL for all the English source')
  ```

  The MLG2923 PARTL contains language-sensitive objects for the Dutch language load:
  
  ```plaintext
  CRTPART PRJ(MLA) GRP(PRODUCTION_DUTCH) TYPE(PARTL) PART(MLG2923)
  TEXT('PARTL for all the Dutch source')
  ```

  The SRC2923 PARTL contains all of the Dutch MRI source code (RPG, DDS, Message files, and so on):
  
  ```plaintext
  CRTPART PRJ(MLA) GRP(PRODUCTION_DUTCH) TYPE(PARTL) PART(SRC2923)
  TEXT('PARTL for all the Dutch MRI source')
  ```
- For each of the preceding part lists, use option 2 from the Work with Parts Using PDM display to add the parts from the following lists to the appropriate PARTL:

<table>
<thead>
<tr>
<th>PARTL</th>
<th>Part Name/Part Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLGOBJ</td>
<td>MLGREFP FILE</td>
</tr>
<tr>
<td></td>
<td>MLGMSTP FILE</td>
</tr>
<tr>
<td></td>
<td>MLGMSTL FILE</td>
</tr>
<tr>
<td></td>
<td>MLGMSTL2 FILE</td>
</tr>
<tr>
<td></td>
<td>MLGMSTL3 FILE</td>
</tr>
<tr>
<td></td>
<td>MLGINQD FILE</td>
</tr>
<tr>
<td></td>
<td>MLGMTND FILE</td>
</tr>
<tr>
<td></td>
<td>MLGMNUD FILE</td>
</tr>
<tr>
<td></td>
<td>MLGNAMD FILE</td>
</tr>
<tr>
<td></td>
<td>MLGNAML FILE</td>
</tr>
<tr>
<td></td>
<td>MLGMNUC PGM</td>
</tr>
<tr>
<td></td>
<td>MLGINQR PGM</td>
</tr>
<tr>
<td></td>
<td>MLGPRODDFN / PRDDFN</td>
</tr>
<tr>
<td></td>
<td>MLGLODCODB / PRDLOD</td>
</tr>
<tr>
<td></td>
<td>RSTLIC110 / PGM RSTLICPGM exit program. This program uses the following parts:</td>
</tr>
<tr>
<td></td>
<td>• USCRTUSRPGR PGM</td>
</tr>
<tr>
<td></td>
<td>• USLOBJRPGR PGM</td>
</tr>
<tr>
<td></td>
<td>• USRTVUSRPG PGM</td>
</tr>
<tr>
<td></td>
<td>• SZRTVLIB1 PGM</td>
</tr>
<tr>
<td></td>
<td>• USCRTUSCMD CMD</td>
</tr>
<tr>
<td></td>
<td>• USLOBJCMD CMD</td>
</tr>
<tr>
<td></td>
<td>• SZRTVLIB CMD</td>
</tr>
<tr>
<td>MLG2924</td>
<td>MLGMTNC PGM</td>
</tr>
<tr>
<td></td>
<td>MLGRPTC PGM</td>
</tr>
<tr>
<td></td>
<td>MLGRPTC2 PGM</td>
</tr>
<tr>
<td></td>
<td>MLGMSGF MSGF</td>
</tr>
<tr>
<td></td>
<td>MLGLODLNGE / PRDLOD LODID(2924)</td>
</tr>
<tr>
<td></td>
<td>MLGPKGMSGF / MSGF</td>
</tr>
<tr>
<td>MLG2923</td>
<td>MLGMTNC PGM</td>
</tr>
<tr>
<td></td>
<td>MLGRPTC PGM</td>
</tr>
<tr>
<td></td>
<td>MLGRPTC2 PGM</td>
</tr>
<tr>
<td></td>
<td>MLGMSGF MSGF</td>
</tr>
<tr>
<td></td>
<td>MLGLODLNGD / PRDLOD LODID(2923)</td>
</tr>
<tr>
<td></td>
<td>MLGPKGMSGF / MSGF</td>
</tr>
<tr>
<td>SRC2924</td>
<td>MLGLOD1E / PRDLOD</td>
</tr>
<tr>
<td></td>
<td>All source parts in PRODUCTION_ENGLISH and PRODUCTION</td>
</tr>
<tr>
<td></td>
<td>It is not very productive to enter all of the source part names one by one into the part list. Use option 2 to edit the part list and use the generic part type and name; see Figure 37 on page 63 as an example.</td>
</tr>
<tr>
<td></td>
<td>Be careful with generic names and types when you use Scan Hierarchy (*YES). The result of such an export can be a lot more source parts than you expect.</td>
</tr>
<tr>
<td>SRC2923</td>
<td>MLGLOD1D / PRDLOD</td>
</tr>
<tr>
<td></td>
<td>All source parts in PRODUCTION_DUTCH and PRODUCTION</td>
</tr>
</tbody>
</table>
6.3.5 Create the Product Libraries and Folders

You need to create the product libraries and folders. In a real environment, the folders also contain documents for things such as product documentation. The product libraries are the target libraries for the EXPPART commands that we do in 6.3.6, “Export the PARTL Parts and Package the Application.”

```bash
CRTLIB MLGCODELIB
CRTLIB MLGDATALIB
CRTLIB MLG2924
CRTLIB MLG2923
CRTLIB MLGSRCLIB
CRTFLR FLR(MLGFLR)
CRTFLR FLR(MLG2924F) INFLR(MLGFLR)
CRTFLR FLR(MLG2923F) INFLR(MLGFLR)
```

6.3.6 Export the PARTL Parts and Package the Application

Use the EXPPART command to export the PARTL parts we have changed in 6.3.4, “Create and Edit the PARTL Parts” on page 61. Specify *LIST on the PARTLOPT parameter to export only the parts listed inside the PARTL part. You must ensure the PARTL part contains a part of type PRDLOD. No PRDLOD part means no packaging.

The EXPPART command recognizes that it has exported a list of parts that contains a part of the type PRDLOD. It extracts information from the part of type PRDLOD and uses that information to set up the packaging information in all other parts that have been exported. If a part of type PRDDFN exists, it extracts information from the part and uses that information to set up the product definition information.

Application Development Manager uses the following steps to export and package the application:

1. All of the parts that are in the PARTL are exported to the library that you specify as TOLIB in the EXPPART command.
2. If there is a PRDDFN part in the PARTL, Application Development Manager uses the CRTPRDDFN command you specified in the PRDDFN part and creates the *PRDDFN object in the library specified in the CRTPRDDFN command and not in the TOLIB library specified in the EXPPART command.
3. Application Development Manager uses the CRTPRDLOD command you specified in the PRDLOD part to create the *PRDLOD object in the library specified in the DVLLIB parameter of the CRTPRDLOD command. The
**PRDLOD** is not created in the TOLIB library specified in the EXPPART command. If the **PRDDFN** does not exist, the **PRDLOD** is not created.

4. To mark the object as belonging to the **PRDLOD**, Application Development Manager changes the object description of every part from the PARTL that was exported using the CHGPRDOBJD command. This includes the **PRDDFN** and **PRDLOD** objects.

5. The PKGPRDOPT is run automatically using the information specified in the **PRDLOD** object. It is run for each **PRDLOD** object created.

---

**Note**

The PKGPRDOPT is run with the parameter REPACKAGE (**YES**) ALWAPICHG (**SAME**). You cannot change this.

---

The steps to export the Mailing List application are:

1. Always export the PARTL with the PRDDFN first. If there is no **PRDDFN** object, the PKGPRDOPT fails. The PARTL MLGOBJ contains the PRDDFN part so that is the one we export first:

   ```
   EXPPART PRJ(MLA) GRP(PRODUCTION) TYPE(PARTL) PART(MLGOBJ) SCAN(*NO) TOLIB(MLGCODELIB) OWNER(QSECOFR)
   ```

   When the export is successful, you see a message in the joblog similar to this: "Product 1MLG001 release V1R1M0 option *BASE load 5001 packaged successfully."

2. Export PARTL MLG2924:

   ```
   EXPPART PRJ(MLA) GRP(PRODUCTION_ENGLISH) TYPE(PARTL) PART(MLG2924) SCAN(*NO) TOLIB(MLG2924) OWNER(QSECOFR)
   ```

   When the export is successful, you see a message in the joblog similar to this: "Product 1MLG001 release V1R1M0 option *BASE load 2924 packaged successfully."

3. Export PARTL MLG2923:

   ```
   EXPPART PRJ(MLA) GRP(PRODUCTION_DUTCH) TYPE(PARTL) PART(MLG2923) SCAN(*NO) TOLIB(MLG2923) OWNER(QSECOFR)
   ```

   When the export is successful, you see a message in the joblog similar to this: "Product 1MLG001 release V1R1M0 option *BASE load 2923 packaged successfully."

4. Export the PARTL SRC2924:

   ```
   EXPPART PRJ(MLA) GRP(PRODUCTION_ENGLISH) TYPE(PARTL) PART(SRC2924) SCAN(*YES) TOLIB(MLGSRCLIB) SRCFILE(SRC2924) OWNER(QSECOFR)
   ```

   When the export is successful, you see a message in the joblog similar to this: "Product 1MLG001 release V1R1M0 option 0001 load 5002 packaged successfully."

5. Export PARTL SRC2923:

   ```
   EXPPART PRJ(MLA) GRP(PRODUCTION_DUTCH) TYPE(PARTL) PART(SRC2923) SCAN(*YES) TOLIB(MLGSRCLIB) SRCFILE(SRC2923) OWNER(QSECOFR)
   ```

   When the export is successful, you see a message in the joblog similar to this: "Product 1MLG001 release V1R1M0 option 0001 load 5002 packaged successfully."
Notes

When we exported the language loads for the *BASE option, we specified SCAN(*NO) on the EXPPART command. This was so that we would be sure to only export the NLS dependant parts. Also, if we were missing an NLS dependant part in the correct group, the EXPPART command would fail, and we would discover our mistake.

When we exported the English and Dutch loads for Option 1, we specified SCAN(*YES) on the EXPPART command. Most of the source parts are not NLS dependant, and therefore reside in group PRODUCTION. We need SCAN(*YES) to pick up the source from the PRODUCTION group since we are doing the export from either PRODUCTION_ENGLISH or PRODUCTION_DUTCH. The output source files, SRC2923 and SRC2924, each have an entire copy of the source for the application.

Do not proceed unless you see the ‘Product packaged successfully’ (CP10CF2) messages in the joblog. If these messages are not found, an error occurred during the packaging process and the error must be corrected and the EXPPART command (or commands) must be run again.

PARTL SRC2924 and PARTL SRC2923 use the same *PRDLOD object. This is not a problem. Application Development Manager recognizes during the export of the second PARTL, that a *PRDLOD object with the same name exists in the library you specified in the DVLLIB parameter and does not create it, but the packaging is still performed. After the export, the object description for the parts exported is changed and the Product is repackaged.

The Mailing List application packaging is now completed.

6.3.7 Save and Install the Licensed Program Product

After the EXPPART command has been run, your product must be saved and restored before it is fully recognized by the system. For instance, if you do not save and restore the product, you do not see the product listed when you execute the DSPSFWRSC command. However, in this “semi-supported” state, some commands such as DLTLICPGM do run against the product. The process to save and restore the product are:

1. Create two save files to which you can save your product:
   
   CRTSAVF FILE(QGPL/MAILLIST1) TEXT(Mailing list Application *BASE)
   CRTSAVF FILE(QGPL/MAILLIST2) TEXT(Mailing list Application Option 1)

2. Run the SAVLICPGM command to save each option:

   SAVLICPGM LICPGM(1MLG001) DEV(*SAVF) OPTION(*BASE) RLS(V1R1M0)
   LNG(*ALL) SAVF(QGPL/MAILLIST1)

   SAVLICPGM LICPGM(1MLG001) DEV(*SAVF) OPTION(1) RLS(V1R1M0)
   LNG(*ALL) SAVF(QGPL/MAILLIST2)

3. You must now run the DLTLICPGM command to remove the application from your system so that it restores correctly in the next step:

   DLTLICPGM LICPGM(1MLG001)

4. Restore the product from the save files to complete the process:
• Restore the *BASE option and the primary language (in this case, we are assuming a primary language of 2924 (English)). All of the objects are restored into library MLGCODELIB, and the empty file shells are copied into MLGDATALIB by the exit program:

RSTLICPGM LICPGM(1MLG001) DEV(*SAVF) OPTION(*BASE) LNG(*PRIMARY) RLS(VIR1MO) SAVF(QGPL/MAILLIST1)

• Restore the 2923 *LNG version for *BASE. All of the objects are restored into library MLG2923:

RSTLICPGM LICPGM(1MLG001) DEV(*SAVF) OPTION(*BASE) RSTOBJ(*LNG) LNG(2923) RLS(VIR1MO) SAVF(QGPL/MAILLIST1)

• Restore option 1:

RSTLICPGM LICPGM(1MLG001) DEV(*SAVF) OPTION(1) RLS(VIR1MO) SAVF(QGPL/MAILLIST2)

The product can also be saved to and restored from tape. If you want to use tape, simply substitute the tape device name (for example, TAP01) into the *DEV parameter on the preceding SAVLICPGM and RSTLICPGM commands.

6.3.7.1 Verify Licensed Program Product Installation

Use the Display Software Resources (DSPSFWRSC) command to verify that the product is registered in the system’s software resource directory.

Use the Check Product Option (CHKPRDOPT) command to verify that there are no errors, for example, objects missing:

CHKPRDOPT PRDID(1MLG001) RLS(VIR1MO) OPTION(*ALL)

Product 1MLG001 release VIR1MO option *BASE load 2924 correctly installed.
Product 1MLG001 release VIR1MO option *BASE load 5001 correctly installed.
Product 1MLG001 release VIR1MO option *BASE load 2923 correctly installed.
Product 1MLG001 release VIR1MO option 0001 load 5002 correctly installed.
No errors detected by CHKPRDOPT.

6.4 Electronic Distribution of Your Product

After you have installed your product on your development system, you can use the SNDPRD command to send your product to target systems. Alternatively, you can send a *SAVF containing your product and receive it and manually restore the product on the target system.

6.4.1 Using the SNDPRD Command

The SNDPRD command allows you to send a product to one or more managed systems and optionally have the product installed automatically. From the central site system, you can monitor and track the results of the SNDPRD command for the managed systems.

Note

For a complete description of the SNDPRD command as well as other pertinent topics on using SM/400 to manage your network from a central system, refer to the Redbook Managing AS/400 Networks with Operations Control Center/400, SG24-4372-01.
An example of how the SNDPRD command can be used to send product 1MLG001 to system SYSNAM1 and immediately installed is shown in Figure 38 on page 67.

Note

If you install your product from the central system, you must ensure that you have no commands in your install exit program that can only run in an interactive environment.

6.4.2 Sending Your Product Using *SAVFs

As an alternative to using the SNDPRD command to distribute and install your product, you can do the following:

1. Use SNDNETF to send the *SAVFs that contain your product to the target system or systems.

2. Sign on to each target system and:
   • Create a set of save files for your product.
   • Receive the net files into the save files.
   • Execute the RSTLICPGM command to install the product.

This process provides the advantage of allowing the RSTLICPGM command to be run interactively, which is necessary if you have commands in your install exit programs that are allowed to run only in an interactive environment. The disadvantage of this process is that it requires more manual intervention than SNDPRD, and requires you to sign on to the target system to receive the *SAVF and execute the RSTLICPGM command.
6.5 Packaging and Installation Tips and Hints

- **Planning for future use:**
  Include a save file object as part of your product. You or your customers might find it useful to save objects in this save file and distribute them with the product after the product is packaged.

  Add dummy exit programs that perform no functions other than those required such as processing the missing objects list but that can be replaced by real programs after the product is packaged. Enhancements to exit program functions can be distributed through PTFs.

  When you create your product definition, you may want to define a few additional options for future additional function. This can save you from having to package a new release of your product. For example, if you have a general ledger application and package it in the *BASE product option, if you had defined an additional option when you created the product definition, you can later package and sell an option to print management reports against the general ledger database without having to create a new release of the product. This technique can extend the amount of time between the need to create a new release of your product.

- In general, your install instructions should state that the RSTLICPGM should be run by QSECOFR or a user profile with equivalent rights as QSECOFR to ensure that all operations in your exit programs can be successfully completed. This may impact the way you configure SM/400 if you plan to use the SNDPRD or INSRMTPRD commands. For more details, refer to the Redbook Managing AS/400 Networks with Operations Control Center/400, SG24-4372-01.

- Exit programs need to handle missing object lists. Pre-operation exit programs should handle errors for *SAVCODE and *SAVLNG operations. Postoperation exit programs should handle errors for *CHKCODE and *CHKLNG operations. Refer to the System Manager/400 Use for more information.

- If your product objects are to be owned by a user profile that is unique to the product, you can use a pre-install exit program to create the user profile.

- Make sure that your exit programs clean up all objects at DLTLICPGM time that were created by your install program at RSTLICPGM time (for example, user profiles, and so on).

- When you install your product on the packaging system, the product goes back to the development libraries, not to the primary and additional product’s libraries, if the product was not deleted before the install.

- Before installing your product in the packaging system, you must delete at least the product’s folders.

- Object type product definition (PRDDFN) and product load cannot be deleted using Application Development ToolSet/400. You must use the Delete Product Definition (DLTPRDDFN) and Delete Product Load (DLTPRDLOD) commands to delete these objects.

- The objects that belong to the code load for an option should not have the same name as the language load objects.
• All of the language loads (object type *PRDLOD) must have the same name and must reside in the development library for the language load at the time you package your application.

• When debugging problems, you can verify that the correct object is being used by displaying the Object Description for the object (or objects) in question:

```
Display Object Description - Service
Library 1 of 1
Object . . . . . . . . . . . . . . . . . : USCRTUSRPG
Library . . . . . . . . . . . . . . . : MLGCODE
Type . . . . . . . . . . . . . . . . . : *PGM
Source file . . . . . . . . . . . . . . : QRPSSRC
Library . . . . . . . . . . . . . . . : MLA.PRD
Member . . . . . . . . . . . . . . . : USCRTUSRPG
Attribute . . . . . . . . . . . . . . : RPG
User-defined attribute . . . . . . . :
Freed . . . . . . . . . . . . . . . . . : NO
Size . . . . . . . . . . . . . . . . . : 15872
Creation date/time . . . . . . . . . : 12/17/93 14:49:25
Source file date/time . . . . . . . . : 12/11/93 13:36:07
System level . . . . . . . . . . . . . : V2R3M0
Compiler . . . . . . . . . . . . . . . : 5738RG1 V2R3M0
Object control level . . . . . . . . :
Changed by program . . . . . . . . . . : YES
More...
Press Enter to continue.
```

```
Display Object Description - Service
Library
Object . . . . . . . . . . . . . . . . . : USCRTUSRPG
Library . . . . . . . . . . . . . . . : MLGCODE
Type . . . . . . . . . . . . . . . . . : *PGM
User modified . . . . . . . . . . . . . : YES
Licensed program . . . . . . . . . . : IMLG001 V1R1M0
PTF number . . . . . . . . . . . . . . : IHP0007
APAR ID . . . . . . . . . . . . . . . :
Text . . . . . . . . . . . . . . . . . : CPP for CRTUSRSPC command
```

• Prototype your packaging and installation process in small scale (few objects).

• Include the commands you use to package and set up your test environment in CL programs.

• Deleting the product using the DLTLICPGM command deletes all of the product’s libraries, even if they include objects that do not belong to the product.

• If your product includes folders, the user packaging the product must be enrolled in the system directory.

6.6 Summary

• The Application Development Manager command EXPPART can be used to invoke the SM/400 packaging commands.

• The Application Development Manager part types PRDDFN and PRDLOD control how the packaging is done at the time the EXPPART command is invoked.
• The SNDPRD command can be used to electronically distribute and optionally install your product to one or more systems.
Chapter 7. Packaging Without the Use of ADM/400

In the previous two chapters, we showed how the Mailing List Application could be enabled for two language versions and packaged using the capabilities of the Application Development Manager and SM/400 products working together.

In this chapter, we show a couple of alternative methods of packaging your product, that are independent of Application Development Manager and use only SM/400 functions. In the process, we show only one language version of the Mailing List Application, which is packaged into one code load. The product identifier is 1MLG002, and this version of the product is used throughout the rest of the book.

### 7.1 Converting Your Application Into a Packaged Product

In the next sections, we discuss how to convert your application into a product. Before proceeding, you should review 6.2, “Converting Your Application Into a Packaged Product” on page 52.

#### 7.1.1 Packaging Process Step by Step

For the following example, we assume that all objects belonging to the Mailing List Application have been placed in library MLGEXP.

Create the MLGEXP library:

```
CRTLIB LIB(MLGEXP) TEXT('export target library for MLG application')
```

**Note**

We use Application Development Manager to get the objects into the MLGEXP library. However, any other source control tool can be used. The key point is that you get all of the objects into one library, called the development library, from which you package the objects. You should be aware of the fact that the development library is deleted during the packaging process so you should view this library as a temporary holding place for your objects.

Check the CRTOUT part list using option 5 from the “Work with Parts using PDM” panel. The part list should not include the module parts and should include the MLGMSGF part. Using options XS,XA, and XR, edit the CRTOUT part list as necessary. The part list should look the same as Figure 39 on page 72.
To get the objects into library MLGEXP, use the EXPPART command:

```
EXPPART PRJ(MLG) GRP(PRODUCTION) TYPE(PARTL) PART(CRTOUT) TOLIB(MLGEXP)
DATA(*YES) OWNER(QSECOFR) AUT(*CHANGE)
```

The application is packaged using SM/400 from library MLGEXP into library MLG. In other words, the MLGEXP library is the development library and the MLG library is the primary library. We also assume that folder MLGFLR exists.

The objects in library MLGEXP are:

- MLGREFP  FILE PF
- MLGMSTP  FILE PF
- MLGMSTL  FILE LF
- MLGMSTL2 FILE LF
- MLGMSTL3 FILE LF
- MLGNAML  FILE LF
- MLGMNUC  PGM CLLE
- MLGINOR  PGM CLLE
- MLGMTNC  PGM CLLE
- MLGRPTC  PGM CLLE
- MLGRPTC2 PGM CLLE
- MLGINQD  FILE DSPF
- MLGTDNC  FILE DSPF
- MLGMNUD  FILE DSPF
- MLGNAMD  FILE DSPF
- MLGMSGF  MSGF

In addition to the application objects, you also need the programs and commands for your install exit program. For more information about the install program, see 7.1.1.3, “Create Exit Programs” on page 74. The list of objects related to the exit program are:
There are also some programs that are used during other examples that we package as part of our product. This is a typical situation; you sometimes have some special product maintenance programs that you ship as part of your product. The maintenance programs are:

- SZRTVLIBPT PGM CL
- USERROR PGM CL
- USQLICOBJD PGM RPG

The additional objects in the preceding two lists are shipped with and become part of your product. Therefore, you need to copy them to the MLGEXP library from library SG244187.1:

CALL SG244187.1/CPY_EXITS

In the next sections, we show how to convert the Mailing List application into a product.

### 7.1.1.1 Create the Packaging Libraries and Folder, the Product Message File, and Add Message Descriptions

```cmd
/*Pgm Name : MLGV1PKGCA */
PGM
*******************************************************************/
/* Create the MLG and MLGDATA libraries and the MLGFLR folder */
*******************************************************************/
CRTLIB LIB(MLG) TEXT('product library for 1MLG002')
MONMSG MSGID(CPF0000)
CRTLIB LIB(MLGDATA) TEXT('data lib for 1MLG002')
MONMSG MSGID(CPF0000)
CRTFLR FLR(MLGFLRA)
MONMSG MSGID(CPF0000)

*******************************************************************/
/* Create a message file for the product. Add the product description as a message in the message file. This product description will be displayed using the DSPSFWRSC command after the product is installed. */
*******************************************************************/
CRTMSGF MSGF(MLGEXP/MLGPKGMSGF) TEXT('MLG code + packaging message file at V1R1M0')
ADDSGD MSGID(MSG0001) MSGF(MLGEXP/MLGPKGMSGF) + MSG('Mailing application *Base')
```
7.1.1.2 Create Product Definition

/*************************************************************************/*/ 
/* */ 
/* Define the Mailing List Application to the AS/400. */ 
/* */ 
/* Create a Product Definition called MLGPRODDFN. */ 
/* - Product ID: 1MLG002 */ 
/* - Product has 1 options: *BASE */ 
/* - Option *BASE has 1 load: Code load */ 
/* - The product’s library can be renamed when the product is installed */ 
/* - Multiple releases of the product can be installed on the same system */ 
/* - Product packaged is at VIRIMO level. */ 
/*************************************************************************/ 

CRTPRDDFN PRDDFN(MLGEXP/MLGPRODDFN) PRDID(1MLG002) +
RLS(V1R1M0) RGSID(*PHONE 123456789) +
MSGF(MLGPKGMSGF) PRDOPT((*BASE MSG0001 +
*ALWDYNNAM *NONE) +
CPYRGTFST(1988) ALWMLTRLS(*YES)

7.1.1.3 Create Exit Programs

We use a post-operation Exit program that runs after the product is restored using the RSTLC1PGM command. The program RSTLC11A (CLP) is defined as a post-operation program (POSTOPRPGM parameter in the CRTPRDLOD command) and its main purpose is to copy the “file shells” from the base option’s primary library (MLG) to the base option’s first additional library where the user’s data files reside after the product is installed (MLGDATA). By copying the files after the product is installed into this library, the files in the production data library are not part of the product.

For a description of the exit program example RSTLC11A, refer to 7.1.1.9, “Sample Exit Program RSTLC11A” on page 76.

7.1.1.4 Create Product Load

/*************************************************************************/*/ 
/* */ 
/* Create the Product Load for option *BASE: */ 
/* */ 
/* - MLGLODCODB: code load. */ 
/* */ 
/*************************************************************************/ 

CRTPRDLOD PRDLOD(MLGLODCODB) PRDID(1MLG002) +
RLS(V1R1M0) OPTION(*BASE) LOCTYPE(*CODE) +
LODID(*CODEDFT) RGSID(*PRDDFN) +
DVLLIB(MLGEXP) PRILIB(MLG) +
PREOPRPGM(*NONE) POSTOPRPGM(RSTLC11A) +
MINTGTRLS(V3R1M0) ADLLIB((MLGDATA))
FLRL((MLGFLRA))
7.1.1.5 Change Product Object Descriptions

```plaintext
CHGPRDOBJD OBJ(MLG/*ALL) OBJTYPE(*ALL)  
PRDID(1MLG002) RLS(V1R1M0) OPTION(*BASE)  
LODID(*CODEDFT)
``` 

7.1.1.6 Package the Product Option

```plaintext
PKGPRDOPT PRDID(1MLG002) OPTION(*BASE)  
ALWAPICHG(*SAME) RLS(V1R1M0) REPACKAGE(*YES)
``` 

At this point, the product is packaged but is not yet registered in the software resource directory on the packaging system. You must save and install the product for the registration to take place.

**Note**

When you run the DTLICPGM command, the MLGEXP library is deleted. This is because this library has been associated with the product in the current product definition and product load. For recovery purposes, you may want to save the library before running the next steps in the procedure if you are following the procedure step-by-step.

7.1.1.7 Save and Install the Product

```plaintext
CRTSAVF FILE(QGPL/MYPROD)
SAVLICPGM LICPGM(1MLG002) DEV(*SAVF) OPTION(*BASE)  
RLS(V1R1M0) SAVF(QGPL/MYPROD)
DLTLICPGM LICPGM(1MLG002)
RSTLICPGM LICPGM(1MLG002) DEV(*SAVF) SAVF(QGPL/MYPROD)
```
To create and install the product on your system, you can run program MLGV1PKGCA:

CALL SG244187.1/MLGV1PKGCA

Use the Check Product Option (CHKPRDOPT) command to verify that there are no errors. Press F10 to see the lower-level messages to be sure there are no errors:

CHKPRDOPT PRDID(1MLG002) RLS(V1R1M0) OPTION(*BASE)
Product 1MLG002 release V1R1M0 option *BASE load 5001 correctly packaged.
No errors detected by CHKPRDOPT.

7.1.1.8 Verify Licensed Program Product Installation

Use the Display Software Resources (DSPSFWRSC) command to verify that the product is registered in the system's software resource directory.

7.1.1.9 Sample Exit Program RSTLIC11A

We use a post-operation Exit program that runs after the product is restored using the RSTLICPGM command. The program RSTLIC11A (CLP) is defined as a post-operation program (POSTOPRPGM parameter in CRTPRDLOD command) and its main purpose is to copy the “file shells” from the base option’s primary library (MLG) to the base option’s first additional library where the user’s data files reside after the product is installed (MLGDATA). By copying the files after the product is installed into this library, the files in the production data library are not part of the product.

Notice that since we allow dynamic naming in our product definition (ALWDYNNAM parameter in the CRTPRDDFN command), we must retrieve the names of the restored libraries in our exit program; we cannot assume that the names are the same as the ones specified in the product loads.

The following summarizes the steps performed by the exit program RSTLIC11A:

1. List all of the physical and logical files in the primary library (MLG or installed library name if the name was overridden during the RSTLICPGM operation).

2. Duplicate the files into the product’s data library (MLGDATA or installed additional library name if the name was overridden during the RSTLICPGM operation).

3. Add product information to the object descriptions (OIR); this is the information about each object that identifies which product it is associated with, such as release level, option, and load identifier. This is done for documentation purposes only since the files duplicated into MLGDATA are not part of the product.

The exit program RSTLIC11A calls other commands and programs to perform these functions. The exit program and all of the commands and programs it calls are part of the product and packaged with it in our example.

Another alternative for creating the files in the data production library is to do the preparation work at the central site where you package your product and
save the files into a Save File in MLG. This save file should be part of the
product and should ship with it. The post-operation exit program should only
restore the files from the SAVF into MLGDATA instead of creating the files and
updating the object description.

**Note**

Refer to Chapter 5 in the *System Manager/400 Use* for a detailed description
of exit program functions and the error messages that should be sent from
exit programs. To keep our sample exit program simple, we are not
including the handling of error situations that should be considered in a real
production environment.
/* PGM Name  RSTLIC11A  */
/* PGM Function: Post-operation EXIT program for RSTLICPGM */
/* This is an example of an exit program for the 1st installation of the product. */
/* Scenario: Only database file descriptions ("file shell") are part of the product and shipped in the product's primary library MLG. */
/* After installing the LPP this EXIT program duplicates product's database files (LF and PF) in the Product's data library, */
/* This way the data base files where the user data is stored are not part of the product  */
/*------------------------------------------------------------------------------------------------*/
/*1) Create the user space that will contain the list of objects (Data Base Files) to be duplicated. */
/* API : USPSRTUS */
/* RPG PGM : USCRTUSRPG */
/* CMD : USCRTUSCMD */
/*------------------------------------------------------------------------------------------------*/
/*2) List all the Objects type *FILE in the the product library in the User Space */
/* API : USPSLOBJ */
/* RPG PGM : USLOBJRPG */
/* CMD : USLOBJCMD */
/*------------------------------------------------------------------------------------------------*/
/*3) Retrieve the Object List stored in the User Space */
/* API : USPSRTVUS */
/* RPG PGM : USRTVUSRPG */
/*------------------------------------------------------------------------------------------------*/
/*4) Get the Installed Library Name for MLGDATA to duplicate the files (SZRTVLIB) and pass control over to USRTVUSRPG */
/* CL PGM : SZRTVLIB1 (CPP) */
/* CMD : SZRTVLIB */
/* API : QSZRTVPR */
/*------------------------------------------------------------------------------------------------*/
/*5) Duplicate the Objects Type *File Special Attribute PF or LF */
/* CRTDUPOBJ */
/*------------------------------------------------------------------------------------------------*/
/*6) Update the object information (for documentation purpose only) to contain product ID and LOAD is */
/* API : QLICOBJD */
/* RPG PGM : USQLICOBJ */
/*------------------------------------------------------------------------------------------------*/

PGM PARM(&FUNCTION &INDICATOR &LNGID &EXITPGMLIB + &CURRENTLIB &CREATELIB &RSTLIB &CURRENTFLR + &CREATEFLR &RSTFLR &CURVRM &RSTVRM + &LMISSOBJ &NRMISSOBJ &LMISSFLR &NRMISSFLR)
/* EXIT pgm PARMS */
DCL VAR(&FUNCTION) TYPE(*CHAR) LEN(10)
DCL VAR(&INDICATOR) TYPE(*CHAR) LEN(10) /* Before or After indicator */
DCL VAR(&LNGID) TYPE(*CHAR) LEN(4)
DCL VAR(&EXITPGMLIB) TYPE(*CHAR) LEN(10)
DCL VAR(&CURRENTLIB) TYPE(*CHAR) LEN(10)
DCL VAR(&CREATELIB) TYPE(*CHAR) LEN(10)
DCL VAR(&RSTLIB) TYPE(*CHAR) LEN(10)
DCL VAR(&CURRENTFLR) TYPE(*CHAR) LEN(12)
DCL VAR(&CREATEFLR) TYPE(*CHAR) LEN(12)
DCL VAR(&RSTFLR) TYPE(*CHAR) LEN(12)
DCL VAR(&CURVRM) TYPE(*CHAR) LEN(6)
DCL VAR(&RSTVRM) TYPE(*CHAR) LEN(6)
DCL VAR(&LMISSOBJ) TYPE(*CHAR) LEN(9980)
DCL VAR(&NRMISSOBJ) TYPE(*CHAR) LEN(4)
DCL VAR(&LMISSFLR) TYPE(*CHAR) LEN(6300)
DCL VAR(&NRMISSFLR) TYPE(*CHAR) LEN(4)
/*------------------------------------------------------------------------------------------------*/
DCL VAR(&LIBIND) TYPE(*CHAR) LEN(1) VALUE(0)
/*------------------------------------------------------------------------------------------------*/

Figure 40 (Part 1 of 2). Sample Exit Program RSTLIC11A
7.2 Packaging Already Installed Applications

In this section, we show a couple of methods that can be used to package an existing application that is already installed on several systems.

We assume that at the starting point in our scenario, the Mailing List application is installed in some of XYZ Company’s branches. To enable PTF support for all the branches, the Mailing List application must be packaged as a product. In this section, we discuss how to package already installed applications without reinstalling the product.

Note

It is very important that you can ensure that the fix level of the application is the same at all sites before attempting this process. The reason you want to package an already-installed application is to enable PTF support, but if you cannot ensure that you are at the same fix level on all systems involved, your PTF process is very difficult to implement.

7.2.1 Process Overview

The objective is to package an application that is already installed in libraries and folders without reinstalling the package. After this process is complete, the application is recognized by the system as a product and can be managed by SM/400.
We recommend that you prototype the process using a few objects with the same structure as the application you are converting (options, loads, folders, and so on). The prototype application does not need to perform any function in order for you to understand and test the packaging process we are describing here. Using an application that contains only a few objects allows you to repeat the process tasks easily and in a short time, and fully understand the steps that you should follow in your environment.

This is a two-phase process:

- **At the central site**, you are extracting the Product Definition (*PRDDFN), Product Load (or Loads) (*PRDLOD), and product Message Files (or Files) (*MSGF) from the packaged product. At the central site, you are using the process described in 6.2, "Converting Your Application Into a Packaged Product" on page 52. You are shipping the object types *PRDDFN, *PRDLOD, and *MSGF obtained from this package to the remote site.

- **At the remote site**, the application is currently installed in libraries and folders and is not recognized by the system as a product. It, therefore, cannot be managed by SM/400 from the central site system. You are restoring the object types *PRDDFN, *PRDLOD, and *MSGF received from the central site and updating the object description of the objects in the libraries to include product information. This converts the already installed application into a product and enables it for management by SM/400.

In our scenario, the library structure of the Mailing List application already installed on the remote systems is:

```
<table>
<thead>
<tr>
<th>Base application objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>*PGM</td>
</tr>
<tr>
<td>+ File Shells</td>
</tr>
<tr>
<td>+ DSPF</td>
</tr>
<tr>
<td>+ MSGF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>*FILE</td>
</tr>
<tr>
<td>(PF, LF)</td>
</tr>
</tbody>
</table>

MLG

MLGDATA

<table>
<thead>
<tr>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>- FLR</td>
</tr>
</tbody>
</table>

MLGFLRA
```

*Figure 41. Already Installed Mailing List application Library/Object and Folder Structure*
Note

It is assumed that the file shells do not exist in the MLG library, and that the application was originally installed with the files only in the MLGDATA library. The file shells are added to the MLG library by this process. This is necessary to enable the PTF process for the application's files.

We describe two ways to package already-installed applications. The end result of each method is the same. You should decide which one is more appropriate in your environment.

The following restriction applies when packaging already-installed applications:

- Dynamic Naming is not supported. The primary and additional library names at the remote site must have the same names as the libraries and folders at the central site where the product was originally packaged.

If you have multiple language loads (not applicable in our example), the language product load (*PRDLOD object type) must be saved at a central site system with the same primary language as the remote site system to which the product is restored.

7.2.2 Method I

7.2.2.1 Central Site Tasks

The primary objective at the Central Site is to extract the Product Definition, Product Load, and Message File from the packaged and installed application and ship them to the remote site. The following describes step-by-step the tasks to be performed:

1. Package the Mailing List application as a product using the method explained in 7.1.1, "Packaging Process Step by Step" on page 71. The program MLGV1PKGCA carries out the packaging function. Note that you must save and restore the product (SAVLICPGM and RSTLICPGM) for the successful installation of the product at the Central site system.

2. Check that the option has been successfully packaged:

   CHKPRDOPT PRDID(MLG002) RLS(V1R1M0) LODID(5001)
   No errors detected by CHKPRDOPT.
   Press PF10 to look at the lower-level messages.

3. Verify that the product has been successfully registered in the system:

   - DSPSFWRSC

   IMLG002 *BASE *CODE Mailing application *Base
   At this point we have created the *PRDDFN, *PRDLODs, and *MSGF that we need to install on the remote site.

4. Save the product definition (MLGPRODDFN), the product loads (MLGLODCODB, MLGLODLNGB), product message file (MLGPKGMSGF), and file shells (MLGREFP, MLGMSTP, MLGMSTL, MLGMSTL2, MLGMSTL3, MLGNAML) to tape or to a SAVF (for electronic distribution). For example:
Notes

For this process to work properly, the object type *PRDLOD for a language load (for example, 2924) must be saved on a system with the same primary language as the remote site system where it is restored. In this example, the product exists as one load; the preceding statement applies if you are supporting multiple language versions of your product.

The file shells were saved in the preceding library because it is assumed that they currently do not exist in the MLG library at the target site.

5. Distribute the tape or SAVF to the remote site.

7.2.2.2 Remote Site Tasks

The Mailing List application is already installed in the remote site as depicted in Figure 41 on page 80. At this point, the application is not registered as a product and, therefore, is not yet enabled for maintenance using PTF support.

Perform the following tasks at the remote site:

1. Restore the *PRDDFN, *PRDLODs, *MSGF, and file shells saved at the central site into the library that is the option’s primary library (MLG in our example):

   RSTOBJ OBJ(*ALL) SAVLIB(MLG) DEV(*SAVF) SAVF(QTEMP/MLGMIGRPKG)

2. Synchronize the internal product table and index. This step is only required because we are not using the “orthodox” method to package a product:

   Call QSZRECOV

   Note

   QSZRECOV rebuilds the product directory.

   At this point, the product is registered in the system as a product. To verify it:

   - DSPSFWRSC
   1MLG002  *BASE  *CODE  Mailing application  *Base

3. Change the product’s object descriptions to include product information:

   Product ID
   Release level
   Product option
   Load ID.

   For example:

   CHGPRDOBJD OBJ(MLG/*ALL) OBJTYPE(*PGM) PRDID(1MLG002) + RLS(V1R1M0) OPTION(*BASE) LODID(5001)
If your product consists of multiple libraries, instead of manually changing the object descriptions as previously shown, you can write a utility program to retrieve the names of the objects that belong to one option and load and change the object descriptions by adding the product’s information using the QSZRTVPR and QLICOBJD APIs. The sample program SZRTVCL1 can be used as a guide.

4. Verify that the product is installed correctly:

    CHKPRDOPT PRDID(1MLG002) RLS(V1R1M0) LODID(5001)

    Press F10 to look at the job log messages to make sure the product option has been checked successfully.

7.2.3 Method II

7.2.3.1 Central Site Tasks

1. Package the Mailing List application as a product as explained in 7.1.1, “Packaging Process Step by Step” on page 71. The program MLGV1PKGCA carries out the packaging function. Note that you must save and restore the product (SAVLICPGM and RSTLICPGM) for the successful installation of the product at the Central site system.

2. Check that the option has been successfully packaged:

    CHKPRDOPT PRDID(1MLG002) RLS(V1R1M0) LODID(*ALL)

    No errors detected by CHKPRDOPT.

    Press PF10 to look at the lower-level messages.

3. Verify that the product has been successfully registered in the system:

    DSPSFWRSC

    1MLGOO2 *BASE *CODE Mailing application *Base

    At this point, we have created the *PRDDFN, *PRDLODs, and *MSGF that we need to install on the remote site.

4. Create a library to hold the objects to be shipped to the remote site and a Save File to save the library:

    /*Pgm Name : MLGMIGRPKG */
    /*******************************************************************/
    /* Create the migration library that will contain the objects used */
    /* to package the application */
    /*******************************************************************/

    START:

    START:

    CRTLIB

    LIB(MLGMIGRPKG) /* Library created at the central site to contain objects that will be sent to remote site to package already installed app. */

    /*

    CR TSAVF

    FILE(MLGMIGRPKG/MLGMIGRPKG) TEXT(*BLANK) /* SAVF to save library MLGMIGRPKG */

5. Move the object types *MSGF, *PRDDFN, and *PRDLODs to the migration library previously created:
Move all objects that are needed to perform the migration at remote site into MLGMIGRPKG.

- MOVOBJ OBJ(MLG/MLGPKGMSGF) OBJTYPE(*MSGF) + TOLIB(MLGMIGRPKG) /* Product Message file */
- MOVOBJ OBJ(MLG/MLGPRODDFN) OBJTYPE(*PRDDFN) + TOLIB(MLGMIGRPKG) /* Product Definition */
- MOVOBJ OBJ(MLG/MLGLODCODB) OBJTYPE(*PRDLOD) + TOLIB(MLGMIGRPKG) /* Product lode for + *code */

**Note**

At this point, you receive the following messages:

Object MLGPKGMSGF in MLG type *MSGF moved to library MLGMIGRPKG.
**Product 1MLG002 no longer properly installed.**

Object MLGPRODDFN in MLG type *PRDDFN moved to library MLGMIGRPKG.
**Product 1MLG002 no longer properly installed.**

Object MLGLODCODB in MLG type *PRDLOD moved to library MLGMIGRPKG.
**Product 1MLG002 no longer properly installed.**

If you had generated PTFs against your product, you also receive the following message.

PTF information could not be saved or restored.

6. Duplicate the file shells into the migration library. For example:

```plaintext
CRTDUPOBJ OBJ(MLGREFP) FROMLIB(MLG) OBJTYPE(*FILE) TOLIB(MLGMIGRPKG)
CRTDUPOBJ OBJ(MLGMSTP) FROMLIB(MLG) OBJTYPE(*FILE) TOLIB(MLGMIGRPKG)
CRTDUPOBJ OBJ(MLGMSTL) FROMLIB(MLG) OBJTYPE(*FILE) TOLIB(MLGMIGRPKG)
CRTDUPOBJ OBJ(MLGNAML) FROMLIB(MLG) OBJTYPE(*FILE) TOLIB(MLGMIGRPKG)
```

7. Save the migration library (MLGMIGRPKG) to Save File or tape for distribution to the remote site:

```plaintext
SAVLIB LIB(MLGMIGRPKG) DEV(*SAVF) + SAVF(MLGMIGRPKG/MLGMIGRPKG)
```

8. Restore the object types *PRDDFN, *PRDLOD, and *MSGF to the original product’s library and check the product option to verify the product is correctly installed:

```plaintext
RSTLIB SAVLIB(MLGMIGRPKG) DEV(*SAVF) SAVF(MLGMIGRPKG/MLGMIGRPKG) + RSTLIB(MLG)
```

The file shells may not restore, just verify that the *MSGF, *PRDDFN, and *PRDLOD objects restore correctly.

```plaintext
CHKPRDOPT PRDID(1MLG002) RLS(V1R1M0) OPTION(*BASE)
Product 1MLG002 release V1R1M0 option *BASE load 5001 correctly installed.
No errors detected by CHKPRDOPT.
```

9. Send the *SAVF to the remote site.

10. Delete the migration library:
7.2.3.2 Remote Site Tasks

The Mailing List application is already installed in the remote site as depicted in Figure 41 on page 80. At this point, the application is not registered as a product and, therefore, is not yet enabled for maintenance using PTF support.

Perform the following tasks at the remote site:

1. Create a save file:
   
   ```
   CRTSAVF QTEMP/MLGMIGRPKG
   ```

2. Receive the network file into the *SAVF just created.

3. Restore the *PRDDFN, *PRDLODs, *MSGF, and file shells saved at the central site into the libraries that become the option’s primary libraries (MLG in our example):

   ```
   RSTLIB SAVLIB(MLGMIGRPKG) DEV(*SAVF) SAVF(QTEMP/MLGMIGRPKG) RSTLIB(MLG)
   ```

4. Change the product’s object descriptions to include product information:

   ```
   Product ID
   Release level
   Product option
   Load ID.
   ```

   For example:

   ```
   CHGPRDOBJD OBJ(MLG/*ALL) OBJTYPE(*ALL) PRDID(1MLG002) + RLS(V1R1M0) OPTION(*BASE) LODID(5001)
   ```

   Refer to Section 7.2.2.2, “Remote Site Tasks” on page 82 for guidelines on how to automate this step for more complex products.

5. Verify that the product option is correctly installed:

   ```
   CHKPRDOPT PRDID(1MLG002) RLS(V1R1M0) OPTION(*BASE)
   ```

   Press F10 to look at the lower-level messages.

   ```
   Product 1MLG001 release V1R1M0 option *BASE load 5001 correctly installed.
   No errors detected by CHKPRDOPT.
   ```

   At this point, the product is registered in the system as a product. To verify it:

   ```
   - DSPSFWRSC
   ```

6. Output:

   ```
   IMLG002 *BASE *CODE Mailing application *Base
   ```

7.2.4 Summary for Packaging Already Installed Applications

7.2.4.1 Method I Pros and Cons

The advantage of Method I is that the process used to extract the object types *PRDDFN, *PRDLOD, and *MSGF (SAVOBJ) does not affect the product installed at the central site.

The disadvantage is that the program, QSZRECOV, that must be run at the remote site to rebuild the product directory, is a long-running program and can...
take several minutes, depending on the number of libraries and products on the system.

7.2.4.2 Method II Pros and Cons
The advantage of this method is that it is not necessary to run QSZRECOV at the remote site. CHKPRDOPT is all you need to run after restoring the object types *PRDDFN, *PRDLOD, and *MSGF and exit programs.

The disadvantage of Method II is that the process to extract the object types *PRDDFN, *PRDLOD, and *MSGF (MOVOBJ) affects the product installed at the central site, even when it is fairly simple to move the objects back to the original library once they have been shipped to the remote system.

7.3 Summary

- You do not need Application Development Manager to package your application. If you are using another method of source control, you can continue to use it and still package your applications using SM/400.
- You can package your application even after it is installed on multiple systems, thus enabling your application for PTF support.
- We recommend using Application Development Manager and SM/400 together as a total solution, but if you have a large investment in a current process, you can package your products as a first step toward migrating to the integrated Application Development Manager and SM/400.
Chapter 8. Servicing Your Products With the AS/400 PTF Process

In this chapter, we cover the product maintenance cycle.

We give a general description of the AS/400 PTF process and how you can implement the process with Application Development Manager and SM/400.

As part of the software maintenance cycle, we use the service provider and service requester capabilities of SM/400 to provide user application problem management and PTF support from a central AS/400 system to a remote AS/400 system.

Before proceeding with the next sections of this chapter, you should review Chapter 1, Setting Up for System Manager/400 and Chapter 7, Servicing Your Products Using System Manager/400 of the System Manager/400 Use manual.

8.1 AS/400 PTF Process Overview

The following are the identified requirements for the XYZ Company maintenance process:

- The ability to create a solution or fix (PTF) for the problem.
- The ability to distribute the solution or fix for the problem.
- The ability to apply the fix to an application.
- The ability to remove the fix from an application.
- The ability to track the fix for a problem.
- The ability to use the same process for IBM and non-IBM products.

SM/400 provides a seamless maintenance process for IBM and non-IBM products with the following attributes:

- Problem logging and tracking support
- PTF packaging support
- PTF electronic distribution
- PTF tracking
- PTF installation

8.1.1 What is a PTF?

An AS/400 Program Temporary Fix (PTF) is a collection of objects that is logically grouped to fix a product’s problem or provide enhancements to the product’s functions. The PTF objects are saved in a save file when the PTF is created. The PTF objects can be replacement objects that replace existing product objects, new objects added to the product, or temporary objects.

A PTF cover letter is documentation about the problem the PTF corrects or a description of the enhancement the PTF provides. A PTF cover letter can be created for each national language version supported by the AS/400 system. Each cover letter is stored in a file member in QGPL/QAPZCOVER.

Two types of relationships can exist between PTFs for your products:
A Prerequisite relationship is when one PTF requires that another PTF also be applied. Prerequisite PTFs are specified when the PTF is created. Prerequisite PTFs must exist within the same product.

A superseded PTF is one that has been replaced by another PTF. If a new PTF has one or more PTF objects in common with an existing PTF, the new PTF supersedes the previous one, and all of the objects of the superseded PTF are included in (or copied to) the new PTF; this is done automatically by the system.

When discussing PTF distribution, it is helpful to understand the concepts of service provider and service requester in terms of the PTF process.

A service provider is a system in an AS/400 network that has been configured to provide PTF service to the other systems in the network. PTFs can be distributed from this provider to the other systems.

A service requester is a system in an AS/400 network that has been configured to be allowed to request maintenance (PTFs) from the service provider.

The relationships between service requester and service provider, and the methods used to configure systems as either a requester or a provider are discussed in detail in Chapter 1 of the System Manager/400 Use.

Management of the PTF process is just one of many functions that can be done using SM/400.

For detailed information on how to use all of the various change management functions of SM/400, refer to System Manager/400 Use. For information on how to manage changes for IBM Licensed Program Products, refer to AS/400 System Startup and Problem Handling.

8.2 Change Scenario

The change team of XYZ Company finds a minor usability problem after the Mailing List application is shipped. The Mailing List application main menu provides a submit of workload to the batch subsystem with option 3 and 4 but does not provide an option to check on the status of the submitted jobs. The change team adds a new menu option 6, “Work with submitted Jobs” to the Mailing List application main menu.

This change is shown in Figure 42 on page 89.
8.3 Develop a PTF Using ADM/400

The following steps are needed to develop our first fix:

1. Add a maintenance hierarchy to the project.
2. Check out the parts.
3. Change and build the parts.
4. Test the fix.
5. Package the parts as a PTF.
6. Test the PTF.

8.3.1 Maintenance Project Hierarchy

In our existing project hierarchy, it is not possible to change parts in the PRODUCTION group without overwriting the original ones. We need to add new groups to the hierarchy that was originally set up; the result is shown in Figure 43 on page 90.

In order to start with a fresh copy of the initial code, we create a new project called “MLG”

CALL SG244187.1/CRTADMEMVG
8.3.2 Determine What Parts Are Involved In this Change

The following parts need to be changed for the implementation of option 6 on the main menu:

1. The display file MLGMNUD to show option 6.
2. The program MLGMNUC to execute the command WRKSBMJOB.
3. The message file MLGMSGF that has a message added with the text: “6. Work with submitted Jobs”
For documentation and change management purposes, we use a PARTL to assist in controlling the PTF process. Because the first character of the PTF ID must be numeric and Application Development Manager requires the first character to be alphabetic, our naming convention for PTF PARTL parts is $PPPnnnnnnn$, where “$PPP$” is the Project ID and “$nnnnnnn$” is the PTF number.

**Note:** Application Development Manager allows you to have PARTL within PARTL; this can be useful for prerequisite PTF’s.

Make MAINT DEVELOPMENT the specified group to work with and create PARTL MLG1MP0020 using the following command:

```
Create Part (CRTPART)
Type choices, press Enter.
Project . . . . . . . . . . . . > MLG  Name
Group . . . . . . . . . . . . > MAINT DEVELOPMENT
Type . . . . . . . . . . . . > mlg1mp0020  Name, *GENERATE
Part . . . . . . . . . . . . > *NONE
Language . . . . . . > *DFT
Prompt create command . . . > *NO
Promote code . . . . . . > *NONE
Source file . . . . . . > *TYPE
Part list . . . . . . > *NONE
Text description . . . > 'PTF 1MP0020 partlist'
```

**Figure 44. Create a Part List for PTF Control**

Add the three determined parts to be changed to the PARTL. See Appendix B, “Utility programs” on page 257 for more information on using some utility programs delivered with this redbook that makes life easier in working with PARTL parts.

After implementing these utilities, you can use “XS” to select the PARTL and “XA” to add the parts to the PARTL.

The PARTL should look similar to this:
### 8.3.3 Check Out the Part (or Parts)

The parts affected by the change are in PARTL MLG1MP0020. Using the "XP" option from the utilities against the PARTL checks out all of the parts into the MAINT_DEVELOPMENT group.

![Figure 46. Check Out the Parts From the PARTL](image)

Press F5 to refresh the display to see that you have actually checked the parts out.

### 8.3.4 Change and Build the Part (or Parts)

Make the following changes to the parts:

- Add a message with Msg ID MSG9999 to the message file MLGMSGF. The text should be:
  
  "6. Work with submitted Jobs"

- Change the source of the display file MLGMNUD; replace it with the source member of the same name in the source file SG244187.1/QSRCCH7.
- Change the source of the program MLGMNUC; replace it with the source member of the same name in the source file SG244187.1/QSRCCH7.

Build the parts with the following command using the PARTL MLG1MP0020; also use the same PARTL as the output PARTL so it can add all of the build parts to it.

![Build Part (BLDPART)](image)

Figure 47. Build the Parts Using a PARTL for Input and Output

Display the MLG1MP0020 PARTL and check if all the parts that should have been built are in the PARTL. Review also the BLDPART report.

### 8.3.5 PTF Documentation

A description of the PTF and installation requirements are included in a cover letter that we create for the PTF.

![Create Part (CRTPART)](image)

Figure 48. Create the PTF Cover Letter
Note

When a part list is mentioned on the CRTPART command, the part is automatically added to that part list.

Copy the contents of the source member as mentioned in Figure 49 into this created TXTSRC part.

Srcfile: SG244187.1/QSRCCH7  Member: CVLMP0020

0001.00
0002.00 PTF number: 1MP0020
0003.00
0004.00 Description: PTF 1MP0020 adds a missing option to the Mailing List application Main Menu. Option 6, “Work with submitted Jobs”, provides extra support for the submits used in option 3 and 4.
0005.00
0006.00
0007.00
0008.00
0009.00
0010.00 Object(s) affected: MLGMSGF message file
0011.00 MLGMNUD display file
0012.00 MLGMNUC program
0013.00
0014.00 Apply PTF instructions:
0015.00
0016.00 1. Ensure no one is using the Mailing application Main Menu.
0017.00
0018.00 2. Load and apply the 1MP0020 PTF using option 11 from the Work with PTF (WRKPTF) display.

Figure 49. Example TXTSRC Cover Letter

Instead of using TXTSRC parts, the user can also create a user-defined part type for a PTF cover letter. This keeps it distinct from the other TXTSRC parts (allowing PDM subsetting, QRYPART subsetting, and so on). The following two commands are all it takes to define the new part type:

ADDADMTYPE TYPE(PTFTEXT) SYSTYPE(*MBR) DFTSRCF(QPTFSRC)
ADDADMLANG TYPE(PTFTEXT) LANG(*NONE)

8.4 Testing the Fix

In Application Development Manager, usually the testing of a fix is done by the programmer in the MAINT_DEVELOPMENT group. If you are confident that the fix is OK, you can promote the fix to the next level in the hierarchy (MAINT_TEST). The “Team leader” performs another test before the fix is ready for promotion to the MAINT.PRODUCTION level and export.

The following steps are involved in testing the fix:

1. Run and test the application in the MAINT_DEVELOPMENT group. Make sure that Scan hierarchy is set to Y in your PDM defaults (F18). Use option 45 to set the appropriate LIBL and then CALL MLGMNUC.
2. Promote the parts two levels (one at a time). See 8.4.1, “Promote a PARTL Part” on page 95 for full details.
3. Export the parts. See 8.4.2, "Export the PARTL Part" on page 96 for full details.

8.4.1 Promote a PARTL Part

If you are promoting a PARTL part, the PRMPART command allows you to specify how you want to promote it. If you specify PARTLOPT(*LIST) on the PRMPART command, each part in the PARTL part is promoted. If you specify PARTLOPT(*PART), only the PARTL part is promoted the same as any other part. If you specify PARTLOPT(*BOTH), both the PARTL part and the parts listed inside it are promoted. If the command you are running ends abnormally, it is possible that some parts in it are not processed. To ensure that all of the parts are processed, you should look for any error messages and, if necessary, run the command again.

In our scenario, we use the following command to promote the parts:

```
PRMPART PRJ(MLG) GRP(MAINT_DEVELOPMENT) TYPE(PARTL) PART(MLG1MP0020)
EXTEND(*YES) PARTLOPT(*BOTH) ARCHIVE(*YES)
```

If the promote is successful, you receive the message:

```
7 parts processed, 2 parts not processed
```

![Work with Parts Using PDM](image)

```
Project ............ MLG
Specified group .... MAINT_TEST
Position to ......... Position to type ....

Type options, press Enter.
2=Change 3=Copy 4=Delete 5=Display 6=Print 7=Rename
8=Display information 13=Change information 14=Build 16=Run ...

Opt Part Type Language Group
MLGMNUC CLESRC CLLE MAINT_TEST
MLGMNUC DBSSRC DSPF MAINT_TEST
CVL1MP0020 TXTSRC *NONE MAINT_TEST
MLGMNUC MODULE CLLE MAINT_TEST
MLGMNUG MSGF *NONE MAINT_TEST
MLGMNUD FILE DSPF MAINT_TEST
MLG1MP0020 PARTL *NONE MAINT_TEST

Parameters or command

F3=Exit F4=Prompt F5=Refresh F6=Create
F9=Retrieve F10=Command entry F23=More options F24=More keys
```

Figure 50. Parts After the Promote

The promote of the PARTL is done with EXTEND(*YES) and PARTLOPT(*BOTH), so the object parts related to a source part are promoted as well, such as:

- MLGMNUD display file
- MLGMNUC module

The MLGMNUC program is not promoted and needs to be re-built, this can be done using the following command:

```
BLDPART PRJ(MLG) GRP(MAINT_TEST) TYPE(PARTL) PART(MLG1MP0020)
```
You should see the message:
1 parts built, 0 parts failed, 0 warnings.

Repeat the promote process to get the parts in the MAINT_PRODUCTION group (including the rebuild).

PRMPART PRJ(MLG) GRP(MAINT_TEST) TYPE(PARTL) PART(MLG1MP0020) EXTEND(*YES) PARTLOPT(*BOTH) ARCHIVE(*YES)

BLDPART PRJ(MLG) GRP(MAINT_PRODUCTION) TYPE(PARTL) PART(MLG1MP0020)

ARCHIVE(*YES) is used so any existing source part in the group that we promote a source part to is archived in the archive library before it is replaced with the new source. A maximum of five archive parts is held at any given time.

8.4.2 Export the PARTL Part

For the export, we only need the parts that are part of the PTF package, so with the PARTL utility options, we remove (option “XR”) the following parts from the MLG1MP0020 PARTL

1. MLGMNUC, program source
2. MLGMNUD, display file source
3. MLGMNUC, module

Finally the PARTL should look the same as the following example:

```
Display Physical File Member
File ........ : MLG1MP0020 Library ........ : MLG.MNTP
Member ........ : QALYPRTL Record ........ : 3
Control ........ Column ........ : 1
Find ...........
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
MSGF MLGMSGF Mailing list message file for MLS support
FILE MLGMNUD Mailing list menu
PGM MLGMNUC Mailing list menu program
TXTSRC CVL1MP0020PTF Imp00020 Cover letter

***** END OF DATA ******
```

Figure 51. Parts to be Exported

Create the PTF library used for the export if it does not already exist with the command:

```
CRTLIB LIB(MLGPTFLIB) TEXT('PTF library for the MLG application')
```
or if it does exist, do a CLRLIB LIB(MLGPTFLIB).

Then export the PTF using the PARTL and the following command:

```
EXPPART PRJ(MLG) GRP(MAINT_PRODUCTION) TYPE(PARTL) PART(MLG1MP0020) SCAN (*NO) TOLIB(MLGPTFLIB) OWNER(QSECOFR)
```

The export to the PTF library is successful if the message “4 parts processed, 0 parts not processed” is displayed. The PTF is now ready for the test and packaging process of SM/400.
8.4.3 Run and Test the Application

To test the fix in the production environment, you should remove any Application Development Manager libraries from your library list using the RMVPRJLIBL command, and add the MLGPTFLIB and the MLG libraries to your library list. You should make sure that the MLGPTFLIB library is ahead of the MLG library in your library list.

You can now run the application by calling the MLGMNUC program:

CALL MLGMNUC

Test that the corrected menu appears, and that option 6 works correctly when invoked. In a real production environment, you also need to perform any regression tests at this time since this is your first opportunity to test your changes in a production environment.

8.5 Create a PTF

Now we are ready to create the PTF.

Whether you use Application Development Manager or another application development tool to develop your fix (PTF objects), the PTF objects eventually reside in a library where the CRTPTF command looks for them to save the objects into the PTF save file. The library specified in the OBJLIB Development Library parameter of CRTPTF is where the PTF objects reside on the system where the PTF is created. The library specified in the parameter OBJLIB Primary Library is where the objects are placed when the PTF is loaded and applied. In our example, the development library is specified as MLGPTFLIB and the primary library is specified as MLG.

The Create PTF (CRTPTF) command saves the PTF objects in a save file in the QGPL library. The save file contains a PTF control object and the PTF objects. The save file name is the PTF identifier preceded by a “Q.” Our PTF numbering system has a standard of “xyyxxxx,” where x is numeric and yy is alphabetic.

Note

The preceding convention was defined for our product based on the requirement that user PTFs must have a numeric in the first position.

Note that the naming convention for IBM PTFs and user PTFs is different. PTFs for IBM products always start with an alphabetic character. User PTFs must always start with a numeric. This can help you to distinguish the different types of PTFs.

For the PTF in our scenario, the PTF ID is 1MP0020. If you use a cover letter to include PTF-related documentation, the cover letter is saved as a member in the
QAPZCOVER source file in the QGPL library. Figure 53 on page 99 illustrates the relationships between the PTF objects in the PTF development library and the PTF once it is created by the following command.

To create the PTF, enter the CRTPTF command as shown in Figure 52.

Figure 52 (Part 1 of 3). The CRTPTF Command

Figure 52 (Part 2 of 3). The CRTPTF Command
Figure 52 (Part 3 of 3). The CRTPTF Command

The entire CRTPTF command is shown in Figure 52 on page 98. Note that you can specify a previous release as the target release (as of OS/400 V3R6) and that you can also specify one or more exit programs, even though we did not use these fields in this example.

Figure 53. Relationships Between Library and Objects When Creating a PTF

8.6 Testing Your PTF

Before you distribute or release a PTF, you should test it in a maintenance testing environment. You may want to make a copy of the PTF using the Copy PTF (CPYPTF) command. Figure 54 on page 100 illustrates the flow that takes place when you perform the PTF operations available in the AS/400 PTF process:

1. Load PTF.
2. Apply PTF temporarily/permanently.
Remove PTF temporarily/permanently.
Delete PTF.

Understanding the AS/400 PTF process helps you design your test cases.

Notes
1. CHG0BJS represents the fixed objects (PTF objects).
2. "OLDOBJJS" represents the original (probably defective) objects.
3. QPZRxxxxxx represents the name of object created or renamed by the various PTF operations. xxxxxx is a sequential numeric number assigned when the load, apply, or remove operations are performed.

Figure 54. Status of Objects During the Various PTF Operations

See Chapter 7. Servicing Your Products Using SM/400 of the System Manager/400 Use for a description of the test cases required in testing a PTF.
Prerequisite PTFs can be specified on the Create PTF (CRTPTF) command. A prerequisite PTF must exist on the system when creating a PTF with prerequisites specified.

8.7 Loading and Applying the PTF

In the testing process, you have undoubtedly loaded, applied, and removed the PTF, perhaps multiple times. When all testing is complete, you should load and apply the PTF one final time.

LODPTF LICPGM(1MLG002) DEV(*SAVF) SELECT(1MP0020) SAVF(QGPL/Q1MP0020)

APYPTF LICPGM(1MLG002) RLS(V1R1M0) SELECT(1MP0020)

Note

Note that we applied the PTF temporarily. This is normally the case; you apply the PTF *TEMP, and after a sufficient time apply it *PERM. You want to eventually get all PTFs applied permanently so any temporary objects, or objects archived by the PTF process, get deleted.

8.8 Release the PTF

When the PTF is fully tested on the central system, you need to use the RLSPTF command. The RLSPTF command makes the PTF available so that remote systems can order it, or the central system can send it:

RLSPTF PTF(1MP0020) LICPGM(1MLG002) RLS(V1R1M0)

8.9 Distributing the PTF

If you want to send the PTF to other sites, you can use the SNDPTF command:

SNDPTF PTFID((1MP0020 1MLG002 V1R1M0)) DESTSRVRQS(*ALL) CHECK(*NO)

Note

In the SNDPTF example, we have set DESTSRVRQS to *ALL to send the PTF to all systems that are listed as service requesters, as seen in the WRKSRVRQS display. We specified CHECK(*NO) to unconditionally send the PTF. *YES first checks with each service requester to see if the PTF is required by that system. There is also an APY parameter that is defaulted to *NONE (not shown in the preceding example), but can be set to automatically apply the PTF either temporarily or permanently on each service requester.

You can also send the PTF save file to remote systems with the SNDNETF command, or save the PTF save file to tape using the CPYPTF command.
8.10 PTF Process Tips and Hints

- Certain object types cannot be included in a PTF. These include object types that cannot be saved, restored, renamed, moved, duplicated or deleted as needed by the PTF operations. These types of objects can be created or deleted using PTF exit programs.

- Prior to OS/400 version V3R2 and V3R6, save file objects should not be included in a PTF. The information contained in a save file is lost when the PTF is created because the create duplicate object operation does not duplicate information in the save file. Work was done at V3R2, V3R6 and V3R7 to eliminate this restriction on the PTF process, and as of these releases, save files can be included.

- One PTF cannot span multiple product options and loads. All of the PTF objects must belong to the same product option, product load, and library.

- As of V3R1M0 of OS/400, you can specify exit programs that run before as well as after the apply or remove of the PTF (or you can specify both apply and remove). The choices for the Run Option that you can specify on the EXITPGM parameter on the CRTPTF command are:
  - *APPLY
  - *REMOVE
  - *BOTH (apply and remove)
  - *PREAPY
  - *PRERMV
  - *PREBTH (pre-apply and pre-remove)

- All of the PTF objects must be installed in the same library (OBJLIB, Primary Library parameter in the CRTPTF command).

- The PTF exit programs can be part of the PTF or can be shipped with the product and called by the PTF process. If the PTF exit program is part of the PTF, you must specify the *PTF value on the EXITPGM keyword for the Type field. Shipping the same exit program in two PTFs for the same product causes one PTF to supersede the other one. Avoid this by including the PTF exit program when the product is originally packaged, and then specify the *OBJLIST value on the EXITPGM keyword for the Type field. Otherwise, rename the program every time you ship a new PTF that uses it.

- You can create temporary PTF objects, that is, objects that are deleted when the PTF is permanently applied. To specify a temporary PTF object, use “QPZ1” as the first four characters of the object name. For example, if you want to ship a data area with some control information in it to be used by a PTF exit program, you can ship it as a temporary object. This method allows you to keep your production library cleaned up.

- Prerequisite PTFs must be applied before the dependent PTF.

- Prerequisite and dependent PTFs must belong to the same product option or the prerequisite PTF must belong to the *BASE product option.

- Prerequisite PTFs are moved to the same status as the dependent PTF being applied.

- If one PTF supersedes previous PTFs (because it has at least one object in common with them), the other PTF objects, symptom string, and exit
programs in the superseded PTFs are copied to the new PTF and the exit programs run in Last Installed First Run order.

For example:

PTF A has exit program x.
PTF B has exit programs y and z, and supersedes PTF A.

When PTF B is applied or removed, the order of execution of the exit programs is:
1. Program y runs.
2. Program z runs.
3. Program x runs.

• If, during the install of a PTF, an exit program fails for any reason that is not monitored with specific messages, the PTF goes into a "Damaged" status. **Do not use the DLTPTF command for a PTF in Damaged status.** The DLTPTF command deletes the PTF save file that you need to recover from the situation. The correct sequence of operations is to load the PTF again from the save file, remove the PTF, and then delete the PTF.

• When you run the CRTPTF command, all of the PTF objects become owned by the user ID running the command. For example, we create a data area called TEST1 in library QTEMP and change the ownership of this data area to XYZ, and then sign on to the system as user ID ABCDEF and run the CRTPTF command as follows:

```
CRTPTF PTF(1MP0098) LICPGM(1MLG002) RLS(V1R1M0) PTOBJ((TEST1 *DTAARA)) OBJLIB(QTEMP MLG) TGTRLS(V3R1M0)
```

After the CRTPTF command runs successfully, we display the contents of the PTF save file, and see:

```
Opt Object       Type   Attribute   Owner  Size
PCCR           *DTAARA  QSYS       4096
TEST1          *DTAARA  ABCEDF    4096
```

Make sure the PTF objects are owned by a valid user ID on the system where the PTF will be loaded and applied. A PTF will not be installed if the owning user profile does not exist on the system you are installing the PTF on.

A way to assure that the PTF objects will be owned by the right user ID is to submit the CRTPTF command to batch using the SBMJOB command, and specify the proper user ID in the USER parameter:

```
SBMJOB CMD(CRTPTF PTF(1MP0098) LICPGM(1MLG002) RLS(V1R1M0) PTOBJ((TEST1 *DTAARA)) OBJLIB(QTEMP MLG) TGTRLS(V3R1M0)) USER(XYZ)
```

One limitation that you may still encounter is that you cannot specify certain system user profiles such as QSECOFR as the user parameter on the SBMJOB command. If you need to have objects in your PTF owned by one of these profiles, you either have to sign on as that profile to run the CRTPTF command, or change the object ownership in a PTF exit program.

• As of OS/400 Version 3 Release 2, a TGTRLS parameter was added to the CRTPTF command so that you can specify that the PTF can be loaded on a previous release of OS/400. If the system on which you are creating a PTF is at OS/400 Version 3 Release 1 and you need to support your application on lower-level systems, you need to follow the procedure described in 11.4, "Servicing Your Product with OS/400 *PRV Support." on page 155.
8.11 Summary

- The PTF process allows you to generate and track fixes to your product.
- You should have a maintenance hierarchy in your Application Development Manager product for fixes.
- You should test not only the fix you are making, but the PTF apply and remove processes as well.
- Use the CRTPTF command to package fixes. From V3R2 and up, the CRTPTF command supports a TGTRLS parameter.
- You should always include a cover letter with your PTF, created as a TXTSRC part in Application Development Manager.
Chapter 9. PTF to Database Files

In this chapter, we discuss some special considerations that must be taken into account when developing PTF for database files. The use of Application Dictionary Services to perform impact analysis is illustrated in 13.1.3, "ADM/400 and ADS/400" on page 172.

9.1 Scenario Description

A user at a remote site reports a problem with the field size of zip code. The Mailing List application manual describes the zip code as a numeric field of six digits in length. In the Mailing List application programs, all of the zip codes are five-digit numeric fields. XYZ Company developers confirm that the zip code should be a six-digit numeric field. Program fixes are developed by XYZ Company at the central site and electronically distributed to the remote sites. Before developing the fix, the developers perform an impact analysis study to determine which objects must be modified. No NLS is used in this scenario, and the scenario continues with the current situation of the previous chapter.

A PTF exit program is used to handle the database file fix.

9.2 Special Considerations for Database File PTFs

Fixes to database files are more complex than fixes to other object types due to the fact that physical files contain user data (as opposed to no data or application data only) and, therefore, additional steps must be included in the PTF process.

In the following list, we summarize some characteristics of SM/400 and the OS/400 PTF process and explain how these features influenced our design for packaging our application and database file PTF process.

- The installation of a new release of a product replaces all of the objects in the old release.

  As explained in 6.1, "Mailing List Application Packaging Library Structure" on page 49, only the "file shells" (empty files) are part of the product and shipped in the primary product library. The database files that contain the user data are created after the product is installed and are not part of the product. Therefore, when a new release of the product is installed, all of the "file shells" are replaced (regardless of whether the file format changed in the new release or not). A post-installation exit program is responsible for identifying changes in files and propagating those changes to the files in the user data library (saving the data before the change and restoring it afterwards).

  By not making the files that contain production data part of the product, we avoid the extra work and time of saving and restoring data and recreating access paths if the file format has not changed.

- When a PTF is loaded and applied, the old objects are replaced by the PTF objects. The PTF exit programs are called when a PTF is temporarily applied, permanently applied, temporarily removed, or permanently removed. Exit programs are called pre- and after the apply or remove operation.
• All of the objects for a PTF must share the same product, option, load ID, and library.

Usually changes to a physical file format implies changes to other object types, such as programs or display files. Ideally, all of the objects affected by the fix should be part of a single PTF.

Backup strategies determine high frequency (daily) database file saves and lower frequency (weekly or even monthly) code saves. For the sake of simplicity, usually database files are stored in “data libraries” that are saved daily separate from the code library saved less frequently.

By placing the “file shells” in the same library as the code objects (programs, display files, printer files, data areas, and so on), it is possible to have only one PTF that contains all of the objects affected by the fix.

• Dynamic naming is the ability of the product option to adapt to library and folder names as specified on the Restore Licensed Program (RSTLICPGM) command. If the product option tolerates libraries with names other than those specified in the Create Product Load (CRTPRDLOD) command, the user can override the product library names at installation time.

In our scenario, we allow dynamic naming in the product definition and ship an empty data library as an additional library that is part of the product. This is the library where the customer data is stored in files that are created by a post-operation exit program that runs after the RSTLICPGM command is executed.

By providing the library with the product and allowing dynamic naming, we can keep track of the library name even if the user decides to rename the library at installation time by calling an API.

---

**Dynamic Naming Considerations**

Throughout the document, you see several examples of packaging with or without dynamic naming. What are the pro’s and con’s:

**PRO (dynamic)**

• Gives a lot of flexibility for the customer to choose the name of the libraries.
• Never a collision with existing names.

**CON (dynamic)**

• Harder to maintain for the software provider.
• Less recognition of the application library names by remote support.
• Cannot use a version convention in the naming of releases.
• Cannot use qualified library names in the application.

One of the other issues in this respect is discussed in 10.3, “Managing Multiple Data Libraries” on page 136, but if you keep the production data libraries out of the packaging, they become dynamic by means of the support that is provided in the application. The application should control the naming convention for the data libraries.

The following sections describe step-by-step the process of developing and creating a PTF that involves changes to a field reference file and physical file (plus dependent objects) using Application Development Manager and SM/400.
Our starting point in the process is the list of objects to be changed that we obtained following the impact analysis study.

### 9.3 Impact Analysis

Until now the changes in our Mailing List application were only minor. Changing one message ID in a message file and rebuilding the display file is not very complicated. Things start to get more complicated if we change the MLZIP field in the field reference file MLGREFP. When we plan to make such a change, it is good to know in advance what objects are affected by this change. With Application Development Manager, we can do impact analysis as follows:

1. Check out the MLGREFP part to your development group.

2. Make the part *STALE just by making the change or pretending to make a change to the MLGREFP part, or you can make a part stale by using the CHGPARTINF command (PDM option 13).

3. Use the BLDPART command with the following parameters:

   ```
   BLDPART PRJ(MLG) GRP(DEVELOPMENT) TYPE(DDSSRC) PART(MLGREFP) SCOPE(*EXTENDED) BLDMODE(*RPTONLY) SAVLST(*YES)
   ```
From the report in Figure 55, you can see which parts are affected by the change of MLGREFP. You do not know what kind of a relationship these parts have with the change in the field reference file. Other questions you can ask:

- Is it only recompiled to overcome a level check?
- Is the field MLZIP really used in this recompiled part?
- Is there any need to change my logic in the affected programs?

Application Development Manager cannot give you answers to these questions. You can use the Build Report printed by Application Development Manager and perform the impact analysis or you can use Application Dictionary Services.
9.4 Summary of Impact Analysis

The result of the work done in 13.1.3, “ADM/400 and ADS/400” on page 172 that discusses the relationship between Application Dictionary Services and Application Development Manager and how they can cooperate is that the following objects are affected:

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Part Type</th>
<th>Source Change</th>
<th>Source Change Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLGREFP</td>
<td>DDSSRC</td>
<td>Yes</td>
<td>Field MLZIP to length 6.0</td>
</tr>
<tr>
<td>MLGMTND</td>
<td>DDSSRC</td>
<td>Yes</td>
<td>Line 108, change MSGID MSG0086 to MSG0901</td>
</tr>
<tr>
<td>MLGMSGF</td>
<td>MSGF</td>
<td>Yes</td>
<td>Add MSGID MSG0901, “Numeric 6.0”</td>
</tr>
<tr>
<td>MLGRPRTR</td>
<td>RPGLESRC</td>
<td>Yes</td>
<td>Line 39, change pos 71 to 72</td>
</tr>
</tbody>
</table>

All of the objects in a single PTF must share the same product, product option, product load ID, and library. More than one PTF is required if the objects involved in the fix do not meet these conditions. Since we do not have NLS versions in our application, we only need to create one PTF. Otherwise, a PTF for each language is required. The identification for our PTF is 1MP0021.

9.5 Developing a Fix Using Application Development Manager/400

The following steps are used to develop the fix to enlarge the zip code field in the Mailing List application:

1. Check out the parts to the MAINT_DEVELOPMENT group as mentioned in Table 1.
2. Change the source in the MAINT_DEVELOPMENT group.
3. Build all related parts.
4. Create Save files to be used by the PTF exit program.
5. Create a PTF exit program to handle the database file changes.
6. Create a PTF cover letter.
7. Promote and test the FIX.
8. Export PTF objects to the PTF development library.
9. Export the DDS source specifications with the file formats for the logical files that need to be recreated when changing the physical file affected by the fix.
10. Create the PTF.

9.5.1 Check Out the Parts to the MAINT_DEVELOPMENT Group

Following the impact analysis results, check out the following parts that need to be changed to the MAINT_DEVELOPMENT group.

Create a PARTL called MLG1MP0021 and using the Utility options XS and XA, add the following parts to it.
1. MLGMTND display file source
2. MLGREFP field reference physical file source
3. MLGMSGF message file.
4. MLGRPTR Mailing list one line report per name

Then using the XP user option against the PARTL, you can check out all of the parts to the MAINT_DEVELOPMENT environment. The result should look similar to the following display:

![Work with Parts Using PDM](image)

**Figure 56. Result After Checking Out the Parts**

### 9.5.2 Change the Source in the MAINT_DEVELOPMENT Group

Based on the impact analysis result, complete the following changes. The completely changed source can be copied from the SG244187.1/QSRCCH8 source file stored under the same member name as the original ones.

- Add the message for the new message ID MSG0901.

![Add Message Description Display](image)

**Figure 57. Add Message Description Display**
- Enlarge the zip code field size from 5.0 to 6.0 in the MLGREFP source member. 2

<table>
<thead>
<tr>
<th>Columns . . . :</th>
<th>1 80</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEU=&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMT PF ......A.......T.Name++++++RLen++TDpB......Functions++++++++++++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0019.00 A</td>
<td>VALUES('AL' 'AK' 'AZ')</td>
<td></td>
</tr>
<tr>
<td>0020.00 A</td>
<td>'AR' 'CA' 'CO' 'CT' +</td>
<td></td>
</tr>
<tr>
<td>0021.00 A</td>
<td>'DE' 'DC' 'FL' 'GA' +</td>
<td></td>
</tr>
<tr>
<td>0022.00 A</td>
<td>'HI' 'ID' 'IL' 'IN' +</td>
<td></td>
</tr>
<tr>
<td>0023.00 A</td>
<td>'IA' 'KS' 'KY' 'LA' +</td>
<td></td>
</tr>
<tr>
<td>0024.00 A</td>
<td>'ME' 'MD' 'MA' 'MI' +</td>
<td></td>
</tr>
<tr>
<td>0025.00 A</td>
<td>'MN' 'MS' 'MO' 'MT' +</td>
<td></td>
</tr>
<tr>
<td>0026.00 A</td>
<td>'NE' 'NV' 'NH' 'NJ' +</td>
<td></td>
</tr>
<tr>
<td>0027.00 A</td>
<td>'NM' 'NY' 'NC' 'ND' +</td>
<td></td>
</tr>
<tr>
<td>0028.00 A</td>
<td>'OH' 'OK' 'OR' 'PA' +</td>
<td></td>
</tr>
<tr>
<td>0029.00 A</td>
<td>'RI' 'SC' 'SD' 'TN' +</td>
<td></td>
</tr>
<tr>
<td>0030.00 A</td>
<td>'TX' 'UT' 'VT' 'VA' +</td>
<td></td>
</tr>
<tr>
<td>0031.00 A</td>
<td>'WA' 'WV' 'WI' 'WY')</td>
<td></td>
</tr>
<tr>
<td>0032.00 A</td>
<td>The ZIP code field has changed for 5 numeric to 6 numeric</td>
<td></td>
</tr>
<tr>
<td>0033.00 A</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>0034.00 A</td>
<td>MLZIP 2 6 0 COLHDG ('ZIP' 'code')</td>
<td></td>
</tr>
<tr>
<td>0035.00 A</td>
<td>EDTCD(X)</td>
<td></td>
</tr>
</tbody>
</table>

F3=Exit  F5=Refresh  F9=Retrieve  F10=Cursor  F11=Toggle  F12=Cancel
F16=Repeat find  F24=More keys

Figure 58. Modify Zip Code with SEU

- Change the message ID for zip code in the MLGMTND source member. The message ID is changed from MSG0086 to MSG0901. 3
• Change the output position in the print layout to keep the field header synchronized with the MLZIP field.  

Figure 60. Change the Report Output Position
9.5.3 Build All Related Parts

Application Development Manager offers very useful facilities for parts building. Instead of building the changed parts one-by-one and building their dependencies as well, you can use Application Development Manager to generate all the new parts based on the parts that you have included in the PARTL. Specify the following parameters in the Build Part (BLDPART) command:

- "EXTENDED" in the "Scope of Build" parameter.
- MLG1MP0021 in the "Part list" parameter.

```
Build Part (BLDPART)

Type choices, press Enter.

Project . . . . . . . . . . . . > MLG Name
Group . . . . . . . . . . . . > MAINT_DEVELOPMENT
Type . . . . . . . . . . . . > PARTL Name, *generic*, *ALL
Part . . . . . . . . . . . . > MLG1MP0021 Name, *generic*, *ALL
Language . . . . . . . . . . > *ALL Name, *generic*, *ALL
Search path . . . . . . . . > *DFT *DFT, name
Scope of build . . . . . . . > *extended *NORMAL, *LIMITED...
Force build . . . . . . . . > *NO *NO, *YES
Build mode . . . . . . . . > *COND *COND, *UNCOND, *RPTONL
Save list . . . . . . . . . . > *NO *NO, *YES, *DLT
Perform bind step . . . . . > *YES *YES, *NO
Part list . . . . . . . . . . > mlg1mp0021 *NONE, *PRV, name
```

F3=Exit  F4=Prompt  F5=Refresh  F12=Cancel  F13=How to use this disp
F24=More keys

Figure 61. Building Parts Using ADM/400 and a PARTL

Check the build report in Figure 62 on page 114 and compare it with the report in Figure 55 on page 108. Notice that the column "Reason for Building" is much more meaningful now.
9.5.4 Create the PTF Exit Program

As we explained earlier, the files that are part of our 1MLG002 product do not contain user data and we refer to them as "file shells". The data files are created in the product’s additional library MLGDATA by a post-operation exit program after the product is restored using the RSTLICPGM command. The PTF process applies the changes directly to the product “shell” files and it is the PTF exit program’s responsibility to propagate these changes to the files that contain customer data.

Because in our scenario there are logical files defined over the physical files that we are changing with PTF 1MP0021, these logical files also need to be re-created. Logical files cannot be PTF objects in the PTF that fixes the physical
file because if the PTF is temporarily applied and then removed, the remove operation fails. (The PTF process attempts to delete physical files before deleting the dependent logical files and, therefore, logical files should not be included as PTF objects.) Our solution to this problem is to include a source file as a PTF object:

- QPZ1LF0021 contains DDS source for “new” format logical files.

Notice the source file is a PTF temporary object (it has a “QPZ1” prefix) and, therefore, is not renamed during the PTF temporarily remove operation. Refer to 9.8, “Export the Logical File Format Source Members” on page 123 for more information on PTF temporary objects.

In our scenario, PTF 1MP0021 contains all of the objects that must be replaced as a result of changing the MLZIP field. All of the objects are handled by the PTF process “business as usual”. So is the field reference file MLGREFP that we are replacing in MLG to keep it updated. During the remainder of this section, we focus on changing the format of the physical file MLGMSTP in the data library and re-creating the dependent logical files.

In order to get a better understanding of all the different places that an EXIT program can be called by the PTF process, we introduce a MATRIXFORM where you can make notes as well. (A blank form is in the example library SG244187.1, sourcefile QSRCCH8.) Using this form helps you break up the process into smaller pieces, thus making each part of the problem easier to deal with. Figure 63 explains the following:

- **1** This column gives the status that the PTF process is in and the action that need to be taken.
  
  For instance: the first two rows (pair) show that the status of the PTF is LOADED and the action to perform is TEMPORARY APPLY. We have a pair, since there can be a Pre and an after action. The number 05 and 01 refer to the “status and action” code supplied to the EXIT program by the PTF process. See the System API Reference manual for the PTF interface to exit programs.

- **2** Describe in this column all of the work that should be done for the Application code and the file “shells” in the application library (MLG) per row, meaning all of the work that is not done by the “business as usual” PTF process.

- **3** Describe in this column all the work that needs to be done for the Production data, in our case the MLGDATA library.

More considerations on the production data libraries are discussed in 9.10.1, “The PTF Process and Production Data” on page 125.
Figure 63. MATRIXFORM Describing the EXIT Program Options

Besides all of the actions that are described in this example of the MATRIXFORM, some other common actions are done by every EXIT program, so our program QPZ1MP0021 performs the following steps:

1. Retrieves the installed library names for product primary library (MLG) and product additional library (MLGDATA).

   The program SZRTVLIBPT that uses the QSZRTVPR API is called to retrieve the installed library names.

   Notice that the PTF library passed to the PTF exit program as a parameter contains the installed name of the library where the PTF objects were placed by the PTF process (MLG in our scenario). We also need the installed name of the additional library where the files that contain the customer data reside (MLGDATA in our scenario).

2. In Figure 64 on page 117, you find a part of the EXIT program that describes all of the actions that are necessary to make this exit program.

   More information on the minimum requirements and parameter list interfaces for exit programs is found in 12.1, “Example of a Minimum Exit Program (PTF) and Its Parameter List” on page 159.
Figure 64 (Part 1 of 3). Actions Taken By the EXIT Program
/* Save all the new created Logicals */
SAVOBJ OBJ(MLGMSTL MLGMSTL2 MLGMSTL3 MLGNAML) +
LIB(&INSPRI) DEV(*SAVF) +
SAVF(&INSPRI/QPZ1SN0021) +
GOTO CMDLBL(LABELCHECK) +

ACT_05 Subroutine
- Save the old format of the logical files, just in case
- we need them for a future remove PTF process, which we
- know already will never happen.
- In general this is the place to save whatever is
- necessary, before the PTF apply process replaces objects

ACT_05:
SAVOBJ OBJ(MLGMSTL MLGMSTL2 MLGMSTL3 MLGNAML) +
LIB(&INSPRI) DEV(*SAVF) +
SAVF(&INSPRI/QPZ1SN0021) +
GOTO CMDLBL(LABELCHECK) +

DB-01 Subroutine
- Here we convert a real production database using all
-information from the "file shells".
- We can delete the logical, since we still have save of
-the old logical "file shell" files made in ACT_05.
- Rename the MLGMSTP masterfile, so we preserve the data
-for the copy processes.
- Create a new MLGMSTP from the "file shell" and copy the
-old data into it.
- Restore all the new logical files from the "file shell" +
savefile.

DB_01:
DLTF FILE(&INSADD/MLGREFP) +
CRTRUPOBJ OBJ(MLGREFP) FROMLIB(&INSPRI) OBJTYPE(*FILE) +
TOLIB(&INSADD) +

Delete logicals */
DLTF FILE(&INSADD/MLGMSTL) +
DLTF FILE(&INSADD/MLGMSTL2) +
DLTF FILE(&INSADD/MLGMSTL3) +
DLTF FILE(&INSADD/MLGNAML) +

Rename MLGMSTP */
RNMOBJ OBJ(&INSADD/MLGMSTP) OBJTYPE(*FILE) +
NEWOBJ(BCKMLGMSTP) +

Create new MLGMSTP */
CRTRUPOBJ OBJ(MLGMSTP) FROMLIB(&INSPRI) OBJTYPE(*FILE) +
TOLIB(&INSADD) +

Update the OIR */
CHGVAR VAR(*OBJTYPE) VALUE(*FILE) +
CHGVAR VAR(*OBJLIB) VALUE(*CAT &INSADD) +
CALL PGM(*PTFLIB/USQLICOBJD) PARM(*OBJLIB +
*OBJTYPE *PRDID *RLS *PTFID *PLODID *OPTION) +
CHGVAR VAR(*OBJLIB) VALUE(*CAT &INSADD) +
CALL PGM(*PTFLIB/USQLICOBJD) PARM(*OBJLIB +
*OBJTYPE *PRDID *RLS *PTFID *PLODID *OPTION) +

Copy data from backup MLGMSTP */
CPYF FROMFILE(&INSADD/BCKMLGMSTP) +
TOFILE(&INSADD/MLGMSTP) MBROPT(*REPLACE) +
FMTOPT(*MAP *DROP) +

Restore logicals */
RSTOBJ OBJ(*ALL) SAVLIB(&INSPRI) DEV(*SAVF) +
SAVF(&INSPRI/QPZ1SN0021) RSTLIB(&INSADD) +

GOTO CMDLBL(LABELCHECK) +

Figure 64 (Part 2 of 3). Actions Taken By the EXIT Program
As you have noticed in the program, two temporary objects QPZ1SO0021 and QPZ1SN0021 (both Save files) still need to be created as well as the QPZ1MP0021 exit program itself. Use the following commands:

```
CRTPART PRJ(MLG) GRP(MAINT_DEVELOPMENT) TYPE(FILE) PART(QPZ1SO0021) +
   LANG(SAVF) PARTL(MLG1MP0021) +
   TEXT('Save file for storing the OLD logicals')

CRTPART PRJ(MLG) GRP(MAINT_DEVELOPMENT) TYPE(FILE) PART(QPZ1SN0021) +
   LANG(SAVF) PARTL(MLG1MP0021) +
   TEXT('Save file for storing the NEW logicals')

CRTPART PRJ(MLG) GRP(MAINT_DEVELOPMENT) TYPE(CLPSRC) PART(QPZ1MP0021) +
   LANG(CLP) PARTL(MLG1MP0021) TEXT('PTF 1MP0021 Exit program')
```

Notice that in the CRTPART command the parts are already added to the MLG1MP0021 part list.

Copy the source for the exit program from the member with the same name in source file SG244187.1/QSRCCH8, and create the program with the following command:

```
BLDPART PRJ(MLG) GRP(MAINT_DEVELOPMENT) TYPE(CLPSRC) +
   PART(QPZ1MP0021) LANG(CLP) SCHPTH(*DFT) PARTL(MLG1MP0021)
```

**Notes:**

1. Our PTF exit program is an oversimplified example just to illustrate the main functions that a PTF exit program must perform. You should add error recovery procedures to assure the integrity of customer data and verify that your exit program handles any error conditions.

You should also consider special backup recovery situations. After applying or removing the PTF (and therefore converting the database file), the file should be backed up immediately. If there is a disk crash after the database file has been converted, restoring from the last backup copy may not be good enough.

2. In our example, the PTF exit program QPZ1MP0021 is PTF specific and shipped with the PTF. For this reason, we specify the *PTF value on the EXITPGM keyword for the TYPE field. This PTF exit program is a PTF
temporary object (refer to 9.8, “Export the Logical File Format Source Members” on page 123 for information on temporary objects) and, therefore, it is deleted by the PTF permanent apply operation. If you are using a generic exit program that is used by multiple PTFs, package the program with the product and specify *OBJLIST on the EXITPGM keyword for the TYPE field (sending the same exit program in two PTFs supersedes the first PTF). The programs SZRTVLIBPT (called by QPZ1MP0021 to retrieve the installed product library names) and USQLICOBJD (called by QPZ1MP0021 to update the object descriptions with product and PTF information) are packaged and shipped with the product 1MLG001 and are used by multiple PTF exit programs.

3. You can design your process for database files PTFs in a different way than the one suggested in this chapter. Our objective here is mainly to highlight the differences between PTFs to database files and PTFs to other AS/400 objects types.

More on the subject of production data is discussed in 9.10.1, “The PTF Process and Production Data” on page 125.

9.5.5 Create PTF Cover Letters

Document the fix in the PTF cover letter and include specific installation instructions (for example, the consequences for the data that is converted and the backup strategy). Use F6 to create a part and use the following keywords:

```
Create Part (CRTPART)

Type choices, press Enter.

Project . . . . . . . . . . . . . . PRJ > MLG
Group . . . . . . . . . . . . . . GRP > MAINT_DEVELOPMENT
Type . . . . . . . . . . . . . . . TYPE > TXTSRC
Part . . . . . . . . . . . . . . . . PART > CVL1MP0021
Language . . . . . . . . . . . . LANG *DFT
Prompt create command . . . . PRMPT *NO
Promote code . . . . . . . . . PRMCODE *GRP
Source file . . . . . . . . . . . SRCFILE *TYPE
Part list . . . . . . . . . . . . . . PARTL > MLG1MP0021
Text description . . . . . . . . TEXT > `PTF 1MP0021 Cover letter`
```

Figure 65. Create CVL1MP0021 Cover Letter
9.6 Promote the Fix

Once the PTFs are tested and found to be correct, we promote all of the objects of the MLG1MP0021 part list. Use the EXTEND(*YES) option, so all possible objects are promoted together with the source.
Figure 67. Promote the MLG1MP0021 Part List

In a real production development environment, you go from a development group to a test group and then to the production group. We follow the same route here.

1. First, promote the parts from the MAINT_DEVELOPMENT group to the MAINT_TEST group. Build all of the parts that could not be promoted with:

Figure 68. Rebuild Necessary Parts After the Promote

If required, perform additional testing within the MAINT_TEST group.

2. After additional testing is completed, promote from the MAIN_TEST group to MAIN_PRODUCTION group. Repeat the rebuild process as described in the previous step.

9.7 Export the Fix

Before we start the export, we need to:

1. Clear the MLGPTFLIB library:
   
   CLRLIB MLGPTFLIB

2. Modify the MLG1MP0021 part list so only all the parts for export are in the list.

   The following parts do not need to be exported:
9.8 Export the Logical File Format Source Members

The PTF exit program QPZ1MP0021 re-creates logical files when the PTF is applied. When the 1MP0021 PTF is temporarily applied, the logical files are re-created using the new file format.

Although there is no change in the source members of the logical files in our scenario, our PTF exit program uses the "new" formats to show how changes to the logical file formats can be handled. Even when in our scenario we do not specifically discuss it, a similar approach can be used for PTFs to source files.

The source file that contains the "new" formats is QPZ1LF0021. Notice that the source file is a "PTF temporary object." PTF temporary objects are not renamed by the PTF temporarily remove operation and are deleted by the apply PTF permanent operation. To define a temporary object, give it a name that begins with the characters "QPZ1". The PTF exit program QPZ1MP0021 is also a temporary object and is deleted when the PTF is permanently applied.

The following steps are needed to export the source:

1. Create a Part List part called SRC1MP0021.
2. Edit the SRC1MP0021 Part List part with the Utility options XS and XA until the part list shows:

Display Physical File Member

<table>
<thead>
<tr>
<th>File . . . . .</th>
<th>SRC1MP0021</th>
<th>Library . . . .</th>
<th>MLG.MNTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member . . . .</td>
<td>QALYPRTL</td>
<td>Record . . . .</td>
<td>1</td>
</tr>
<tr>
<td>Control . . . .</td>
<td></td>
<td>Column . . . .</td>
<td>1</td>
</tr>
<tr>
<td>Find . . . . .</td>
<td>*...+....1....+....2....+....3....+....4....+....5....+....6....+....7. .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDSSRC</td>
<td>MLGMLST</td>
<td>Mailing list label printing logical</td>
<td></td>
</tr>
<tr>
<td>DDSSRC</td>
<td>MLGMLST2</td>
<td>Mailing list by state, city, search</td>
<td></td>
</tr>
<tr>
<td>DDSSRC</td>
<td>MLGMLST3</td>
<td>General purpose LF for querying MLGMLSTP</td>
<td></td>
</tr>
<tr>
<td>DDSSRC</td>
<td>MLGMLAML</td>
<td>Mailing list logical by name, state, city</td>
<td></td>
</tr>
</tbody>
</table>

****** END OF DATA *****

Figure 70. SRCMP0021 Part List Part to Export New Source Members

3. Export all four logical file source file members to the QPZ1LF0021 source file:

EXPPART PRJ(MLG) GRP(MAINT DEVELOPMENT) TYPE(PARTL) +
PART(SRC1MP0021) SCAN(*YES) TOLIB(MLGPTFLIB) +
SRCFILE(QPZ1LF0021) OWNER(QSECOFR)

9.9 Create the PTFs

Finally, we are ready to produce the PTF. Compared to the previous PTF, this one has a lot more objects that need to be packaged. It is for that reason that you should look at another utility program example that does a lot of the typing for you. For more information, see Section B.7, "Create a CRTPTF Command String" on page 266.

Use the following Utility command to create this PTFs:

SG244187.1/CRTPTFCMD EXITPGM(QPZ1MP0021) PTFLIB(MLGPTFLIB)

CRTPTF PTF(1MP0021) LICPGM(LICPGM) +
RLS(VRIMO) LODID(5001) +
PTFOBJ((MLGINQ *PGM) (MLGMNC *PGM) +
(MLGRPTC *PGM) (MLGRPTC2 *PGM) +
(QPZ1MP0121 *PGM) (MLGMSGF *MSGF) +
(MLGINQD *FILE) (MLGMLSTP *FILE) +
(MLGMTND *FILE) (MLGSMAN *FILE) +
(MLGREFP *FILE) (QPZ1LF0021 *FILE) +
(QPZ1MNO0021 *FILE) (QPZ1S00021 *FILE)) +
OBJLIB(MLGMLSTL ) +
COVER((MLGMLSTL/QTXXSRC CV1MP0021 2924)) +
EXITPGM((MLGMLSTL/QRZ1MP0002 1 *PREBTH *PTF))

Check the result of the command build and fill in the LICPGM parameter, in this case 1MLG002. Press Enter and the PTF is created.
9.10 Test the PTFs

After the PTF is created, it must be tested before shipment. Since an exit program is involved, all combinations of PTF operations should be tested. See Chapter in the System Manager/400 Use for more information on PTF test cases.

The following key test cases must be checked:

- The problem is fixed by the PTF.
- The OIR for the fixed objects should contain the right PTF number.

With the Utility program described in Section B.8, “Display the LICPGM and PTF Information of an Object” on page 268, you can create a user option PI that shows what the LICPGM and the PTF number is of an object.

- All PTF operations can be performed for all of the different statuses of a PTF without any problem.

Table 2 shows the different test cases that should be performed.

<table>
<thead>
<tr>
<th>PTF Status</th>
<th>PTF Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save file only</td>
<td>Load PTF</td>
</tr>
<tr>
<td></td>
<td>Delete PTF</td>
</tr>
<tr>
<td>Loaded (Not applied)</td>
<td>Temporarily apply</td>
</tr>
<tr>
<td></td>
<td>Permanently remove</td>
</tr>
<tr>
<td>Temporarily Applied</td>
<td>Temporarily remove</td>
</tr>
<tr>
<td></td>
<td>Permanently apply</td>
</tr>
</tbody>
</table>

- The logical files are created for the right physical file in the right library.
- The customer data in the physical files is not affected by the PTF operations.

9.10.1 The PTF Process and Production Data

Keep in mind that the scenario we have discussed so far is only an example. Since most applications have multiple production data libraries, you should make the update of these libraries a separate process to be controlled by the owner of the application and the data.

Use a control file that holds all of the information on the existing production data libraries and their status. The status of all of the libraries should be updated by the PTF exit program so they become invalid after a database change to the “File shells.”

The application should recognize that the selected production database is not in the proper status and cannot be used before it is “migrated”.

The “migration” process should be delivered together with the PTF process and can be a part of the exit program if we only have one data library.

During the migration process, a lot of reasons can cause the process to fail since so many additions can be made to the database definition by the customer:
1. New logicals over existing application data:
   In our example, we delete four logical files before we can delete the
   physical. If more logicals are added by the customer that are not part of the
   application, the delete of the physical fails. (Of course, we can detect which
   logicals and what source they are created from, but it is not the
   responsibility of the software provider to solve all of these problems.)

2. Journaling implemented on the physical files:
   If a physical file is journaled and that physical file is recreated, it ends the
   journaling capability. However, nobody notices this until the customer wants
   to use the journal entries. If you want to solve this problem, you need to
   start the journaling for that file again and make a full save of the file to
   synchronize it with the journal process.

3. Triggers added to the physical files:
   It might seem that triggers are application independent. You just add a
   trigger and then catch an action to be performed. However, it can cause
   some problems to your application if the trigger fails.

   To clarify: a trigger is a user-written program that is associated with a
   database file. Whenever an operation takes place regardless of the interface
   that is changing the data, the trigger program is automatically activated by
   DB2/400 and its logic executed.

4. Constraints added to the physical files:
   Constraints are a real application-dependent implementation. It is hard to
   imagine that they were added by a customer without effects to the
   application.

   To clarify: Referential integrity (constraints) is a set of mechanisms by which
   a database manager enforces some very common integrity rules among
   database files.

5. Altering the database definition:
   Today, it is also possible for the customer to alter a physical file without
   affecting your application. However, if you re-create the file in a PTF
   process, trouble is just around the corner. You undo the customer’s
   (illegal?) change if you PTF the file with the modified original layout.

It should be clear by looking at some of the preceding comments that it is very
important to have a good understanding of the dos and do nots regarding the
database of a software provider application. In general, it is fair to say that
software providers can only be responsible for the database designs that they
implement for their applications. Just imagine how to solve (analyze) a reported
problem when it is caused by an I/O error in a trigger program not designed by
the software provider.

If a customer wants to add something to the database, it should be well
documented so it can be removed before the “migration” starts and,
re-implemented when it is finished. And it should be brought to the attention of
the software provider, if necessary.
9.10.1.1 What’s On the Horizon

After all of the potential problem areas have been discussed, there might be a better solution for users of the V3R7, V3R6 or V3R2 OS/400 release.

From V3R2 and up, we have the ability to alter existing database files, using the SQL or native interface. In SQL, you alter the database by the ALTER command that defines the difference with the existing definition. In native, you have to supply the DDS source with the complete layout, run the CHGPF command, and the system implements the difference.

If we return to our example in this chapter, changing the MLZIP field from five numerics to six numerics, the following scenario is possible for V3R7, V3R6, or V3R2:

1. Impact Analysis and changing the source is still the same.
2. Determining what needs to be shipped in the PTF is different.
3. The Exit program is much simpler.
4. No need to worry about other logicals that might depend on our MLGMSTP physical file.
5. No backup and copy data (it is all done automatically).

The MATRIXFORM should look similar to the following:

<table>
<thead>
<tr>
<th>APPLY</th>
<th>Application Routine (ACT_xx)</th>
<th>Production Database Routine (DB_xx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD 05</td>
<td>1. CHGPF MLGMSTP with new source</td>
<td>1. Replace the MLGREFP file</td>
</tr>
<tr>
<td>Pre-apy TEMP</td>
<td>2. CHGPF MLGMSTP with new source</td>
<td>3. Update OIR of Logicals + MLGMSTP</td>
</tr>
<tr>
<td>LOAD 01</td>
<td>3. Update OIR of Logicals + MLGMSTP</td>
<td></td>
</tr>
<tr>
<td>Apy TEMP</td>
<td>4. CHGPF MLGMSTP with new source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. Update OIR of Logicals + MSTP + REFP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. Replace the MLGREFP file</td>
<td></td>
</tr>
</tbody>
</table>

Note: Certain combinations do not appear on this form, since they do not occur, such as:

If the PTF is in LOAD status and a PERMANENT apply is asked for, the exit program runs twice.

1. LOAD status, pre-apply and apply TEMP
2. APPLIED status, pre-apply and apply PERM

Figure 71. MATRIXFORM Describing the EXIT Program Options
The code in the Exit-program shown in Figure 72 on page 128 is the part that handles the action from the MATRIXFORM. The complete Exit-program is in SG244187.1/QSRCCH8 MBR(QPZ1MP0121).

Figure 72 (Part 1 of 2). Exit Program Action Part for CHGPF Solution
9.10.1.2 How CHGPF Works

This is not a full description of the possibilities and limitations of the CHGPF command, but a more specific view in relation to the PTF process that we have discussed before.

As we have seen before, the control over a database definition might be disturbed by dependencies, added by a customer, and not designed by the software provider. What are the effects when you use CHGPF as a changing mechanism for the database definition, in the area of:

1. Logical files over application data?

   ADDITION of a field:
   - The PF is re-created based on the new DDS. The data is saved and copied into the newly-created file.
   - The LF is re-created if the relation is based on a record format reference so the added field is also present in the new logical file.
   - The LF is not touched if the relation is based on field reference.

   ALTERATION of a field attribute:
• The PF is re-created based on the new DDS. The data is saved and copied into the newly-created file.
• The LF is re-created if the relation is based on a record format reference so the altered field is changed in the new logical file.
• The LF is not touched if based on field reference, neither with nor without reference of the field to the physical file. (This might cause a mapping error to the physical file.)

REMOVAL of a field:
• The PF is re-created based on the new DDS. The data is saved and copied into the newly-created file. A warning message is issued that data will be truncated (C I). Be careful, running in an interactive environment is recommended.
• The LF should be deleted if the relation is based on a record format reference. This can be done in the CHGPF command with keyword DLTDPLF(*YES). Re-creation of the logical files need to be done in some other way.
• The LF is not touched if the relation is based on field reference and the removed field is not part of the LF.

2. Journaling implemented on the physical files?
The physical file retains journaling, but the journaling has been ended and started during the CHGPF process. So for this specific reason, you should make a full save of the file before you start using the file again. Saving the complete database is, from a recovery point of view, the best solution since this file can only be recovered from this save point.

3. Triggers added to the physical files?
The triggers that were implemented for this physical file are retained so you do not need to worry about re-adding that information after the re-creation of the file.

Depending on the structure of the trigger program, this has an influence. The trigger program is always supplied with a parameter list that contains some general information about the trigger and an actual record image before-image as well as after-image. This image is from the physical file, and since the physical file has changed, this image is changed as well. You are able to write programs that can handle this well, since the offset of the start of the record and the length are provided in the parameter list. This needs some sophisticated programming and might not solve all problems. So usually you end up with some re-programming or at least recompiling.

4. Constraints added to the physical files?
The constraints for the physical file are retained if:

• A field belonging to a constraint is not part of a removal process.

5. Altering of the database definition?
In most cases, the previous (illegal) alteration of a physical file is undone, and according to the rules we have seen in the bullet about “Logical files over application data,” it is likely that some of the logical files need to be deleted. Illegal additions act as a removal of fields in the undo process. Logical files need to be re-created in another way.

6. Other advantages:

• Performance implications:
Another benefit of the CHGPF solution is in the performance area (to be precise, in building an access path). If a logical file or physical file has an access path that does not contain any field that was part of the change, this access path is not re-created.

So especially for large databases with large access paths, this can be a tremendous time saver.

- Authority:
  The Ownership, public, and private authority are retained.

- Auditing:
  Object auditing is automatically restarted.

### 9.10.1.3 Conclusion

It is obvious that it is still not easy, nor is it recommended, to change a database within release boundaries since you are still talking about live production data and hardly any chance for a customer to prepare for the change if needed.

We assume that it is sometimes impossible to avoid the addition to existing application databases. Therefore, it is much easier for a customer to prepare for the change if it is at a release boundary when two versions of a release can be operational. This gives you the opportunity to create a new database and migrate all of the data under application control. If customers want to add some of their own definitions back onto the database, they have time to do it and test it all before moving to the new database release.

### 9.10.1.4 An Alternative Scenario

If you want to exercise the scenario with the CHGPF method, you must be aware of the following changes that need to be made to the existing scenario, assuming that you have first exercised the existing scenario:

1. Although the Redbook is intended for V3R7, V3R6 or V3R2 most of it applies to V3R1 as well. This scenario can only run at V3R7, V3R6 or V3R2 since the CHGPF extended command is only supported by that release.

2. First, delete the existing licensed program 1MLG002 by using:

   \[
   \text{DLTLICPGM LICPGM(1MLG002) RLS(V1R1M0)}
   \]

3. Restore the basic licensed program from the earlier exercise and put the first PTF on it as follows:

   \[
   \begin{align*}
   \text{RSTLICPGM LICPGM(1MLG002) DEV(*SAVF) RLS(V1R1M0) + REPLACERLS(*NO) SAVF(QGPL/MYPROD)} \\
   \text{LODPFT LICPGM(1MLG002) SELECT(1MP0020) RLS(V1R1M0)} \\
   \text{APYPTF LICPGM(1MLG002) RLS(V1R1M0) SELECT(1MP0020)}
   \end{align*}
   \]

4. Create a new PARTL part called MLG1MP0121.

5. Create a new PARTL part called QPZ1DB0121.

6. Create a new CLPSRC CLP part called QPZ1MP0121; this contains the new exit program.

7. Create exit program QPZ1MP0121, copy it from the SG214187.1/QSRCCH8 file with a member of the same name, and run the BLDPART command.

8. Create a new TXTSRC *NONE part called CVL1MP0121; this contains the new cover letter.
9. Create cover letter CVL1MP0121. You can copy the letter from SG214187.1/QSRCCH8 MBR(CVL1MP0121).

10. Use the utility options XS andXA to get all of the necessary parts into PARTL MLG1MP0121. The part list finally looks similar to the following:

```
Display Physical File Member
File ........ Library ........
Member ........ Record ........
Control ........ Column ........
Find ........

*...1....2....3....4....5....6....7
MSGF MLGMSGF *PERM Mailing list message file for MLS support
FILE MLGREFP Mailing list field reference file
FILE MLGNTND Mailing maintenance display file
FILE MLGINQD Mailing list inquiry display
FILE MLGNAMD Mailing list name search display
PGM MLGRPTC Print one liner with MLGMSTL2 LF
PGM MLGRPTC2 General purpose query of MLGMSTP
PGM MLGINQR Mailing list inquiry
PGM MLGNTNC Mailing list maintenance
PGM QPZ1MP0121 Exit program for CHGPF Solution
TXTSRC CVL1MP0121 Cover letter for CHGPF solution
```

Figure 73. Part List MLG1MP0121

11. Add the following source entry into the QPZ1DB0121 part list.

```
Display Physical File Member
File ........ Library ........
Member ........ Record ........
Control ........ Column ........
Find ........

*...1....2....3....4....5....6....7..
DDSSRC MLGMSTP Mailing list Master file for CHGPF
****** END OF DATA ******
```

Figure 74. Add Entry to Part List

12. Run the following commands:

```
CLRLIB LIB(MLGPTFLIB)

EXPPART PRJ(MLG) GRP(MAINT_PRODUCTION) TYPE(PARTL) +
PART(MLG1MP0121) TOLIB(MLGPTFLIB) +
OWNER(QSECOFR)

EXPPART PRJ(MLG) GRP(MAINT_PRODUCTION) TYPE(PARTL) +
PART(QPZ1DB0121) SCAN(*YES) TOLIB(MLGPTFLIB) +
SRCFILE(QPZ1DB0121) OWNER(QSECOFR)
```

13. Create the PTF and use the following command:

```
SG244187.1/CRTPTFCMD EXITPGM(QPZ1MP0121) PTFLIB(MLGPTFLIB)
```

Check the result of the command build and fill in the proper LICPGM parameter: 1MLG002.
14. The PTF is built and you should load and apply the PTF.

### 9.10.2 Summary

- Because of the added complexity of handling customer data, it is best to try to limit changes to your database to a release boundary. However, since this is not always possible, this chapter has addressed the additional considerations when making a PTF to one or more files.

- The problems related to changing a database design are ones that need to be addressed, whether you are packaging your application, or have your own packaging strategy. The advantage of using the SM/400 PTF method is that it allows you to break the problem down into smaller, more manageable pieces as shown in the examples in this chapter. Using a form such as MATRIXFORM that was presented can help you to understand what actions have to happen at which part of the PTF process.

- In some situations, you may be able to use the CHGPF command within your PTF process. If you are:
  - Adding a field to a physical file.
  - Altering a physical file attribute and all logical files related to the changed physical have a relation based on record format reference.

  CHGPF can be used as a solution in your PTF.

- Doing a complete impact analysis with a tool such as ADS/400 is a key part of the process when making database changes, whether those changes are implemented through a PTF or at a release boundary.
Chapter 10. Developing and Packaging a New Release of Your Application

This chapter describes the use of Application Development Manager in setting up and packaging the project hierarchy to enable the development of the next release of the Mailing List application.

We discuss special considerations for packaging and installing a new version of a product.

We also discuss considerations for when to quit supporting a given release and some of the issues that this may raise.

10.1 Reasons For a New Release

If you are a commercial software developer, some of the reasons you should consider developing and packaging a new release of your application are:

- Generation of revenue for your business.
- You may want to leverage new function in AS/400 hardware or software to improve your application’s marketability.
- Providing value to your customers for their maintenance agreements.
- Continuity - setting expectations from your customers that they can expect new releases of your product on a regular basis.
- Addition of major enhancements or options to your product to make it marketable.

Whether your application is for sale, or is only supporting your own business, some of the technical reasons you should consider developing and packaging a new release of your application are:

- You may want to leverage new function in AS/400 hardware or software to improve your application’s technical capabilities. For example, you may want to use ILE to improve the performance and design of your application.
- You are adding new or changed function that is too complex or too large for a PTF.
- You are making changes to your database. In general, you should try to limit changes to your file formats to a new release, and not make these kind of changes through a PTF.
- To refresh your PTF cycle. As the number of PTFs for a current release grows, the complexity of managing prerequisites and supersede chains can becomes a problem. You should create new releases often enough to keep this from becoming a problem.
- Size of PTFs. When your PTF supersede chains start becoming lengthy, the size of your PTFs can grow quickly because all objects from superseded PTFs are “dragged” along into your new PTF.
- If you are introducing a new function that could result in a change to your install exit programs, you should only introduce this function at a release boundary, and not through a PTF.
A need to clean up your development environment. After a time, your developer’s Application Development Manager groups for a given release can become quite messy with test objects, and so on unless you have some very strict cleanup policies. This can impact developer productivity over time. By moving to a new release and giving the developers a “clean slate”, you can help improve their productivity.

10.2 Functional Enhancements for V2R1M0 of Mailing List Application

The XYZ Company has decided to create a new release of the Mailing List application.

The following enhancements are added to the Mailing List application:

- License management - we add keyed license compliance.

  Note

  For a detailed description of license management, see 13.5, “License Management Overview” on page 181.

- An empty option for new function that is made available three months after the release of V2R1M0.

- The following PTFs from V1R1M0 are incorporated into V2R1M0 of the Mailing List application:
  - 1MP0020 (Main Menu usability problem)
  - 1MP0021

  The baseline for V2R1M0 release upgrade for existing V1R1M0 customers includes the previous cumulative tape PTF objects up to 1MP0021. That is, all PTFs up to 1MP0021 must be applied before a release update. To ensure that this is the case, you can add code in your install exit program to run prior to RSTLICPGM that checks this requirement. The code has to do a DSPPTF to an outfile and then read the outfile to ensure that the correct PTF level is installed.

  - To use Application Development Manager to do the packaging rather than having a separate packaging program.

10.3 Managing Multiple Data Libraries

In addition to the changes to the Mailing List application listed in the previous section, we also want to change the way the data libraries are handled within the packaged application. The change we want to make is to allow the customer to define multiple data libraries from within the application. We remove the data library name from the product load and instead, define a control file from within the application. The control file lists the names of all of the data libraries that the customer defines. There is a special program shipped with the new version of the product that the customer can use to define new data libraries to the application. When this program is run, the customer is asked to input the name of the new data library. The data library is created by the program and the control file is updated. If the library already exists, an error message is returned to the program.
Note

We do not actually implement the code examples for this enhancement, but are including the discussion of this enhancement to show a workable solution to the real-life problem of how to allow multiple data libraries within your application and the issues of this design as they relate to release upgrades and PTFs to files.

If you want to allow your customers to define multiple data libraries with names that they choose, you should control this within your application as previously discussed and not try to do this by the combination of defining additional libraries within your product and allowing dynamic naming. There are several reasons for this:

• If you name additional data libraries in your product, you are defining a finite number of libraries at product creation time. As soon as one customer has a requirement for one additional library beyond the number you defined, you have to consider a new release of the product.

• If you define the data libraries within your product and do not allow dynamic naming of these libraries, you have locked your customer into a set of library names that may not be meaningful for them.

• If you define the data libraries within your product and allow dynamic naming, you are going to have to do additional coding in subsequent release install exit programs to retrieve the names of the libraries. This is possible through the use of APIs, but can be somewhat confusing to implement and can be easily overlooked at the time the release upgrade exit programs are being written.

If you want to allow multiple data libraries for your application, the best choice is to control it from within your application with the control file and program design, as previously outlined. You still have to consider the additional data libraries when you upgrade to a new release, but the information that you need is readily available in the control file. You can be very specific in your documentation for the application that you will not handle data in any libraries that were not defined by the library setup program. Of course, the control file and the program should be restricted to someone who has special authorities so that the information in the file can be counted on to be reliable at the time when your PTF programs or install programs need to use it.

In terms or our example in this book, we have some additional items to consider since the design of Version 1 of the product did not allow for multiple data libraries. We should probably assume that at least one customer defined additional libraries anyway, as there was nothing within the design of the application to prevent them from doing so. If we want to provide the customer with a way to migrate the data from these additional libraries, then our upgrade process should be to:

• Provide the control file and an update program with the new release, and give the customer install instructions on how to create the new libraries in such a way so that the control file is updated, and run the program to update the files in the new libraries with existing data from the Version 1 libraries.

The control file should at least contain the following information:

1. The name of the Production Data library
2. Creation time stamp

3. The release (VxRyMz) of the application for which the library is valid

4. Last database PTF applied

5. Status:
   
   If you have to go through a conversion process, either being a new release or a DB PTF update, this field can be checked by your application to allow the database to be used.

6. Comments on the status:

   Assuming that your conversion program can give some more information on the type of failure.

7. Last update time stamp

If you need to generate a PTF for a database file, the PTF process should be two-phase:

- Load and apply the PTF to update the file shells in the product library.
- Update the file in the data libraries using a program that is either part of the product or part of the PTF.

By making the update of the data libraries a two-phase process, you eliminate a lot of the complexity in the PTF process, and shift the control of the data update to the data owner.

The data update program should use the control file previously mentioned that holds all of the information on the existing production data libraries and their status. The status of all of the libraries should be updated by a PTF exit program so the application can recognize that the selected production database is not synchronized with the PTF level of the application and, therefore, cannot be used until it is “migrated”.

10.4 Creating the New Version of the Mailing List Application
    
    This section goes through the steps required to create the new version of the Mailing List application.

10.4.1 Version 2 Project Hierarchy
    
    Developing a V2R1M0 of Mailing List application requires a new set of Application Development Manager hierarchy.

    The Maintenance project hierarchy is now changed to support the V2 Development groups.
To create the version 2 environment:

```
CALL SG244187.1/CRTADMV2
```

We added the following new groups to the project hierarchy:

<table>
<thead>
<tr>
<th>Promote Code</th>
<th>Group Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>PRODUCTION_V2</td>
</tr>
<tr>
<td></td>
<td>TEST_V2</td>
</tr>
<tr>
<td></td>
<td>DEVELOPMENT_V2</td>
</tr>
</tbody>
</table>

10.4.2 Developing the Changes Using ADM/400

How to use Application Development Manager is described in detail in:

- Chapter 8, “Servicing Your Products With the AS/400 PTF Process” on page 87.
This section does not describe again how to use Application Development Manager to perform program changes. The key point is to develop and test the programs described in Section 13.5.5, “Modifying Your Programs to Use License Management” on page 188 according to the following project hierarchy:

1. Code and unit test the programming changes initially in the DEVELOPMENT_V2 group.
2. Perform additional functional test, or integrated test, in the TEST_V2 group.
3. Promote to the PRODUCTION_V2 group, ready for exporting to the packaging libraries for a final packaging and installation test.

10.4.3 Export and Package MLG Version 2

The steps to export and package V2R1M0 are:

1. Create, import, and check out the necessary parts into the V2 groups.
2. Change the parts.
3. Add the additional parts’ names from V2R1M0 to the PARTL parts.
4. Export and package V2R1M0.

10.4.4 Create, Import, Check Out the Parts Into the V2 Groups

We are going to purposely shortcut the normal development process by developing all of the parts for this example in PRODUCTION_V2 rather than DEVELOPMENT_V2. You should understand by now how to build and promote parts in your development hierarchy, and we do not want to bore you with extra steps.

The parts needed in PRODUCTION_V2 are:

<table>
<thead>
<tr>
<th>Part Type</th>
<th>Part Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRDDFN</td>
<td>MLGPRODDFN</td>
</tr>
<tr>
<td>PRDLOD</td>
<td>MLGLODCODB</td>
</tr>
<tr>
<td>PARTL</td>
<td>CRTOUT</td>
</tr>
<tr>
<td>MSGF</td>
<td>MLGPKGMSGF</td>
</tr>
<tr>
<td>CLPSRC</td>
<td>LICREQC</td>
</tr>
<tr>
<td>CLPSRC</td>
<td>LICRLSC</td>
</tr>
<tr>
<td>CLLESRC</td>
<td>MLGMNUC</td>
</tr>
<tr>
<td>CLPSRC</td>
<td>RSTLC210</td>
</tr>
</tbody>
</table>

2. Import part MLGPKGMSGF from library MLG. The reason that you need to import the MSGF is so that you do not have to re-create the existing message description.
3. Check out the source for MLGMNUC.
4. Check out the CRTOUT PARTL.

Note: When you import the MLGPKGMSGF, Application Development Manager copies the object, including the object description. Since we imported the MSGF from the existing product, the product information is contained in the object description of the new part. This is not a concern because when the new version of the product is packaged through the EXPPART command; the CHGPRDOBJD command that runs updates the object description for the MSGF with the new release information. The only way to delete packaging information from the object description of an object is to use the QLICOBJD API. Normally,
you do not need to do this but you should be aware of this restriction. The source code USRTVUSRPG in SG244187.1/QSRCCH12 has an embedded example of using the QLICOBJD API.

At this time, group PRODUCTION_V2 should look similar to the example in Figure 76.

We also need some additional programs related either to the exit program RSTLIC210 or to other examples later in the book. The objects are:

- USERROR PGM
- USCRTUSRPG PGM
- USLOBJRPG PGM
- USRTVUSRPG PGM
- SZRTVLIB1 PGM
- USCRTUSCMD CMD
- USLOBJCMD CMD
- SZRTVLIB CMD

To quickly import these objects into the PRODUCTION_V2 group, use the IMP_EXITS program:

```
CALL SG244187.1/IMP_EXITS
```

10.4.5 Edit the Parts in the V2 Groups

The PRDDFN and PRDLODs need to be edited to add the commands. The new Version, Release, Modification level, and preoperation exit program is specified. Use option 2 to change these parts.

- Edit the MLGPKGMSGF Message file:

  Add the following two messages to the packaging message file:

  ```
  MSGID(MSG0002) MSG(‘Mailing list application, *BASE V2R1M0’)
  MSGID(MSG0003) MSG(‘Mailing list application, Option 1, V2R1M0’)
  ```
• Change the product definition in group PRODUCTION_V2. Use member MLGPRODDFN in SG244187.1/QSRCCH9 to update the part.

Columns . . . : 1 71 Edit MLG.PRODV2/QDFNSRC
SEU==> MLGPRODDFN
FMT ** ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
************************************************** Beginning of data **************************************************
0001.00 CRTPRDDFN PRODDFN(MLGV2/MLGPRODDFN) PRDID(1MLG002)+
0002.00 RLS(V2R1M0) RGSID(*PHONE 123456789) +
0003.00 MSGF(MLGPKGMSGF) PRDOPT(*BASE MSG0002 +
0004.00 *NODYNNAM *NONE) (1 MSG0003 *NODYNNAM +
0005.00 *NONE 5002)) CPYRGTFST(1988) +
0006.00 ALWMLTRLS(*YES)
************************************************** End of data **************************************************

F3=Exit F4=Prompt F5=Refresh F9=Retrieve F10=Cursor
F16=Repeat find F17=Repeat change F24=More keys
(C) COPYRIGHT IBM CORP. 1981, 1993.

Figure 77. Change MLGPRODDFN

• Change the PRDLOD parts in PRODUCTION_V2.

Columns . . . : 1 71 Edit MLG.PRODV2/QLODSRC
SEU==> MLGLODCODB
FMT ** ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
************************************************** Beginning of data **************************************************
0001.00 CRTPRDLOD PRDLOD(MLGLODCODB) PRDID(1MLG002) +
0002.00 RLS(V2R1M0) OPTION(*BASE) LODTYPE(*CODE) +
0003.00 LODID(5001) RGSID(*PRDDFN) +
0004.00 DVLLIB(MLGV2) PRILIB(MLG210) +
0005.00 PREOPRPGM(RSTLIC210) PSTOPRPGM(RSTLIC210) +
0006.00 MINTGTRLS(V3R1M0) +
0007.00 FLRL((MLGFLR.210))
************************************************** End of data **************************************************

F3=Exit F4=Prompt F5=Refresh F9=Retrieve F10=Cursor
F16=Repeat find F17=Repeat change F24=More keys
(C) COPYRIGHT IBM CORP. 1981, 1993.

Figure 78. Change MLGLODCODB

Note that we specified a different primary library and folder for this version of the product. We did this so that the two versions of the product can be installed on the target system at the same time. This allows testing of the new version by the customer while they are still running in production on the old version.

We did not allow dynamic naming so the customer is not able to restore the V2R1M0 code library over the V1R1M0 code library. If you want to allow the customer to do this, then you have to allow dynamic naming in the V2R1M0 production definition. Since we are not defining a data library as an additional library for V2R1M0, we do not have to worry about the customer dynamically naming a data library for V2R1M0 over the V1R1M0 data library (if this was the case, you have to handle saving the current data in the section of your exit program that runs before the RSTLICPGM process).
We do not need to create the PRDLOD part for option 1 at this time. It can be created as the code for option 1 is developed.

- Update part LICREQC with source from SG244187.1/QSRCCH9.
- Update part LICRLSC with source from SG244187.1/QSRCCH9.
- Update part RSTLIC210 with source from SG244187.1/QSRCCH9.
- Update part MLGMNUC with source from SG244187.1/QSRCCH9.

10.4.6 Build All of the New or Changed Parts

You need to build all of the new or changed parts. They are:

- LICREQC
- LICRLSC
- RSTLIC210
- MLGMNUC

You can build these parts individually, or you can create a part list to control the build.

When the parts are built correctly, update the CRTOUT part list to add programs LICREQC, LICRLSC, and RSTLIC210. Also add MSGF MLGPKGMSGF, PRDLOD MLGLODCODB, and PRDDFN MLGPRODDFN.

10.5 Packaging the V2R1M0 Release

The packaging development environment is the set of libraries and folders where your objects and documents reside on your development system before you package and save your product. This library and folder structure is not necessarily the same structure your product will have once it is installed in production using the Restore License Program (RSTLICPGM) command. If you use Application Development Manager to develop your software, the export function copies the parts (objects) from Application Development Manager development groups into the packaging development libraries that you specify. We recommend that you use different names for the packaging development libraries and primary and additional libraries where the product is installed. By using different names, you can start working on packaging development (writing exit programs and testing the process for new release installation) on the same system where you have the previous release of the product installed; in fact, you need the previous release of your product installed to test the new release installation scenarios. See 10.6.1, “Sample Exit Program RSTLIC210” on page 145 for more information on RSTLICPGM exit programs.

The process used to package the new version of the product is:

- Create library MLGV2, which is the target library for export:
  
  CRTLIB MLGV2

- Create the MLGFLR.210 folder. Add any necessary documentation into the folder.
  
  CRTFLR FLR(MLGFLR.210)

- Export the contents of the CRTOUT PARTL to MLGV2 to perform the packaging:
  
  EXPPART PRJ(MLG) GRP(PRODUCTION_V2) TYPE(PARTL) PART(CRTOUT) SCAN(*YES) TOLIB(MLGV2) OWNER(QSECOFR) AUT(*CHANGE) TGTRLS(V3R1MO)
Make sure that you display the joblog and find the message:

Product 1MLG002 release V2R1M0 option *BASE load 5001 packaged successfully.

- Create a *SAVF:
  CRTSAVF QTEMP/V2

- Save the newly packaged version:
  SAVLICPGM LICPGM(1MLG002) DEV(*SAVF) RLS(V2R1M0) SAVF(QTEMP/V2)

In our example, the ADDPRDLICI program is run at this time because it was set up in program RSTLIC210 to run *BEFORE *SAVCODE. Look for the confirmation message in your joblog:

License information added for product 1MLG002 V2R1M0 feature 5001 with handle XXXXXXXXXXXXXX.

or use the WRKLICINF command to verify that the product was saved with licensing information.

Note that the Product License Information Handle is included in the message.

- Delete the new version in preparation for the install:
  DLTLCICPGM LICPGM(1MLG002) RLS(V2R1M0)

- Restore the new version:
  RSTLICPGM LICPGM(1MLG002) DEV(*SAVF) RLS(V2R1M0) REPLACERLS(*NO) SAVF(QTEMP/V2)

  **Note**

  We specified REPLACERLS(*NO) on the RSTLICPGM command so that the new release is installed concurrently with the old release.

- Verify correct installation of the product:
  CHKPRDOPT PRODID(1MLG002) RLS(V2R1M0)

  Product 1MLG002 release V2R1M0 option *BASE load 5001 correctly installed.
  Product 1MLG002 release V2R1M0 option 0001 load 5002 only defined.

  Note that option 1 has been defined to the system at this point.

- Use DSPSFWRSC to verify that both versions of the product are installed concurrently:
  DSPSFWRSC

  **Display Software Resources**

<table>
<thead>
<tr>
<th>Resource ID</th>
<th>Option</th>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1MLG002</td>
<td>*BASE</td>
<td>5001</td>
<td>Mailing application *Base</td>
</tr>
<tr>
<td>1MLG002</td>
<td>*BASE</td>
<td>5001</td>
<td>Mailing list application, *BASE V2R1M0</td>
</tr>
</tbody>
</table>

  Press Enter to continue.

  F3=Exit  F11=Display libraries/releases  F12=Cancel
10.6 Installation of V2R1M0 Mailing List Application at XYZ Company

In V1R1M0 of our product 1MLG002, we specified to allow dynamic naming and allow multiple releases in the Create Product Definition (CRTPRDDFN) command. However, in V2R1M0, we have specified no dynamic naming, and have chosen a new primary library, MLG210, for the product. We still allow multiple releases of the product to exist on the system. So, our install process can be somewhat simplified.

CRTPRDDFN PRDDFN(MLGCODELIB/MLGPRODDFN) PRDID(1MLG001) +
RLS(V1R1M0) RGSID(*PHONE 123456789) +
MSGF(MLGPKGMSGF) PRDOPT(*BASE MSG0001 +
*NODYNNAM (2923 2924) 5001) (1 MSG0002 +
*NODYNNAM *NONE 5002)) CPYRGTFST(1988) +
ALWMLTRLS(*YES)

Since the new release of the product is restored into a different library, there is no collision between the new and old version. The new version can be installed and brought up while production work is still going on in V1R1M0. It is assumed that a program is provided with the V2R1M0 version of the product that allows the user to create one or more data libraries for version 2, as discussed in Section 10.3, "Managing Multiple Data Libraries" on page 136. A program to convert existing production data from the V1R1M0 data library to a new data library with new file formats should also be provided as part of the V2R1M0 package.

For the sake of example, we also show how the first data library can be created by the install exit program for V2R1M0. We do not copy any V1R1M0 data into this library at this time.

Figure 80 on page 149 shows how V1R1M0 and V2R1M0 are concurrently installed.

Once the new version of the application has been installed, the data libraries have been created, and any data conversions have been done to move the V1R1M0 production data into one or more of the new data libraries, the procedure to remove V1R1M0 of the product is:

1. Make a full save of the product and data libraries for V1R1M0 of the Mailing List application.
2. Use the DLTLICPGM command to delete V1R1M0 of the product.
3. Delete any additional data libraries that were created for V1R1M0:
   
   DLTLICPGM LICPGM(1MLG002) RLS(V1R1M0)

10.6.1 Sample Exit Program RSTLIC210

The exit program RSTLIC210 in our scenario is a simplified example of how to handle new release installation activities. During the packaging of 1MLG002 V2R1M0, RSTLIC210 was specified as a preoperation and as a post-operation exit program:

The CRTPRODLOD command used:
The following summarizes the steps performed by the exit program RSTLIC210:

1. Prior to SAVLICPGM (function being run is *SAVCODE and the before/after indicator is *BEFORE), run the ADDPRDLICI command.

2. Prior to DLTLICPGM (function begin run is *DLTCODE), the MLGDATA2 data library is deleted.

   **Note:** Remember that your install exit should clean up anything at DLTLICPGM time that it created at RSTLICPGM time.

3. After the RSTLICPGM (function being run is *RSTCODE and the before/after indicator is *AFTER), create a data library called MLGDATA2 and copy the files shells into the data library. Update the object description of the files in the data library to document that they are associated with product 1MLG002, release V2R1M0.

   **Note:** You should refer to the System Manager/400 User for a detailed description of exit program functions and the error messages that should be sent from exit programs. To keep our sample exit program simple, we are not including the handling of error situations that should be considered in a real production environment.

---

```c
/* PGM Name RSTLIC210 */
/* PGM Function: Post-operation EXIT program for RSTLICPGM */
/* This is an example of an exit program for the 1st installation of the product. */
/* Scenario: Only descriptions for data base files ("file shells") are part of the product and shipped in the product primary library MLG. */
/* After installing the LPP this EXIT program duplicates product's data base files (LF and PF) into a data library. */
/* This way the data base files where the user data is stored are not part of the product */
/* Create the user space that will contain the list of objects (Data Base Files) to be duplicated. */
/* API : QUSCRTUS */
/* RPG PGM : USCRTUSRPG */
/* CMD : USCRTUSCMD */
/* List all the Objects type *FILE in the product library in the User Space */
/* API : QUSLOBJ */
/* RPG PGM : USLOBJRPG */
/* CMD : USLOBJCMD */
/* Retrieve the Object List stored in the User Space */
/* API : QUSRVTUS */
/* RPG PGM : USRVTUSRPG */
```

Figure 79 (Part 1 of 3). Sample Exit Program RSTLIC210
/* 4) Duplicate the Objects Type *File Special Attribute PF or LF */
/* */
/* CRTDUPOBJ */
/* */
/* */
/* */
/* */
/* API : QUSRTVUS */
/* */
/* RPG PGM : USRTVUSRPG */
/* */
/* */

PGM PARM(&FUNCTION &INDICATOR &LNGID &EXITPGMLIB + &CURRENTLIB &CREATELIB &RSTLIB &CURRENTFLR + &CREATEFLR &RSTFLR &CURRENTFLR &CREATEFLR + &RSTFLR &LMISSOBJ &NRMISSOBJ &LMISSFLR &NRMISSFLR)
/* EXIT pgm PARMS */

DCL VAR(&FUNCTION) TYPE(*CHAR) LEN(10) /*
Before or After indicator */
DCL VAR(&INDICATOR) TYPE(*CHAR) LEN(10) /* *
DCL VAR(&LNGID) TYPE(*CHAR) LEN(4)
DCL VAR(&EXITPGMLIB) TYPE(*CHAR) LEN(10)
DCL VAR(&CURRENTLIB) TYPE(*CHAR) LEN(10)
DCL VAR(&CREATELIB) TYPE(*CHAR) LEN(10)
DCL VAR(&RSTLIB) TYPE(*CHAR) LEN(10)
DCL VAR(&CURRENTFLR) TYPE(*CHAR) LEN(12)
DCL VAR(&CREATEFLR) TYPE(*CHAR) LEN(12)
DCL VAR(&RSTFLR) TYPE(*CHAR) LEN(12)
DCL VAR(&CURRENTFLR) TYPE(*CHAR) LEN(6)
DCL VAR(&CREATEFLR) TYPE(*CHAR) LEN(6)
DCL VAR(&RSTFLR) TYPE(*CHAR) LEN(6)
DCL VAR(&LMISSOBJ) TYPE(*CHAR) LEN(9980)
DCL VAR(&NRMISSOBJ) TYPE(*CHAR) LEN(4)
DCL VAR(&LMISSFLR) TYPE(*CHAR) LEN(6300)
DCL VAR(&NRMISSFLR) TYPE(*CHAR) LEN(4)
DCL VAR(&LIBIND) TYPE(*CHAR) LEN(1) VALUE('0') /* Product Option Id */
DCL VAR(&OPTION) TYPE(*CHAR) LEN(4) VALUE('0000') /* Product Option Id */
DCL VAR(&LIBIND) TYPE(*CHAR) LEN(1) VALUE('D') /* Product Option Id */
DCL VAR(&OPTION) TYPE(*CHAR) LEN(4) VALUE('0000') /* Product Option Id */
/* Prior to saving the product, add the licensing information */
/* We put this command here to make sure that the FIRST time */
/* the product gets saved on the development system, the */
/* License information will be added. After the product is */
/* installed, this command can not be run, and will result in */
/* one of the messages that we monitor for, so as not to */
/* cause an error. */
/* */
Figure 79 (Part 2 of 3). Sample Exit Program RSTLIC210
IF COND((&FUNCTION *EQ `*SAVCODE`) *AND +
&INDICATOR *EQ `*BEFORE`) THEN(.DO)
ADDPROCLIB PROD(IMLOG02) RLS(V2R1M) +
USGTYPE(`*CONCURRENT`) COMPLIANCE(`*KEYED`) +
DFTUSGLMT(2) LICTRM(`*VERSION`) GRACE(30 *NO)
MONMSG MSGID(CPF9E02 CPF9899 CPA0701)
ENDD

/* If this is a DLTLICPGM operation, delete the MLGDATA2 library */
/* -------------------------------------------------------------------*/
IF COND(&FUNCTION *EQ `*DLTCODE`) THEN(.DO)
DLTLIB LIB(MLGDATA2)
MONMSG MSGID(CPF0000)
ENDD

/*---------------------------------------------------------------*/
/* New Install of Product */
/* or */
/* New Release Install of Product */
/* -------------------------------------------------------------------*/
IF COND(&FUNCTION *NE `*RSTCODE`) THEN(GOTO +
CMDLBL(END)) /* This is not RSTLICPGM +
/* -------------------------------------------------------------------*/
IF COND((&FUNCTION *EQ `*RSTCODE`) *AND +
&INDICATOR *EQ `*AFTER`) THEN(.DO) /* The +
following statements must be executed +
post-operation (`*AFTER installing LPP`) /*
/* First Install or REPLACERLS `*NO in RSTLICPGM +
/* -------------------------------------------------------------------*/
IF COND(&CURRENTLIB *EQ +
&RSTLIB) THEN(.DO) /* &CURRENTLIB = +
1st install &CURRENTLIB *NE +
&RSTLIB: REPLACERLS `*NO */
ADDLIB LIB(AEXITPGMLIB)
MONMSG MSGID(CPF0000) EXEC(CHGVAR VAR(&LIBIND) +
VALUE(`1`) /* Library Already in LIBL */
/* This is the section of code where we create the first */
/* data library from the new files shells shipped with the */
/* product. */
/* -------------------------------------------------------------------*/
CRTLIB LIB(MLGDATA2) TEXT(`Sample data library for +
mailing list V2R1M0`)
/* The call below is an example how you would add the MLGDATA2 */
/* library to the control file of data libraries that are */
/* maintained within the application */
/* The call is commented out, since we have not actually created */
/* the UPDLIBLOG program for this example */
/*
CALL PGM(UPDLIBLOG) PARM(`MLGDATA2`) */
/*
Create User Space */
USCRTUSCMD USRSPEC(QTEMP/SPACE) EXTRAT(NONE) SIZE(1025) +
INZ(O) AUTH(`ALL`) TEXT(`Create User Space +
to store Object List`) +
MONMSG MSGID(CPF0000) /* User Space Already exists */
/* List Files in Product Library */
USLBOJCMC USRSPC(QTEMP/SPACE) FNMTM(OBJL0200) +
OBJECT(&RSTLIB/*ALL OBJTYPE(*FILE) /* +
List all files in product library */
/* Copy the file shells to the data library */
CALL PGM(USRTVUSRPG) PARM(`MLGDATA2` `1MLG002` +
`V2R1M0`) &OPTION `*SO01`) /*
IF COND(&LIBIND *EQ `0`) THEN(RMVLIBLE +
LIB(AEXITPGMLIB)) /* If library wasn't in +
library list before execution of this +
program, remove it */
ENDD
ENDD
END: ENDPGM

Figure 79 (Part 3 of 3). Sample Exit Program RSTLIC210

148 AS/400 Software Life Cycle Management
**Tip**

We have shown several variations of the RSTLICxxx program. All examples have embedded calls to other CL and RPG programs through commands that, in turn, use APIs to do various functions such as retrieving installed library names, listing the files in the project into a data area, and so forth. We did this so that if you encounter a situation where you need to use the APIs, you have an example. You should realize that there are more simple ways to code these exits. For example, in RSTLIC210, we can simply do a series of CPYF commands into the data library since there are not many files involved. After each CPYF command, we can do a CHGPRDOBJD command to change the OIR for the new file just created. The point here is that you do not need to make your exit programs overly complicated; do what makes the most sense for your situation.

---

**Figure 80. Installation of a New Release in a Parallel Environment**
10.7 Dropping Support For an Old Release

As you develop new releases for your product, you need to consider whether or not to discontinue support for older releases of the product. Some of the things that must be considered when deciding whether or not to continue supporting old releases are:

- The cost of maintenance of the old release.
- The amount of dual maintenance involved in supporting multiple releases. This can become troublesome and can also lead to errors, even with automatic notification facilities such as those offered in Application Development Manager. If you are making a fix in a old release, and are changing the same part in a development environment for a new release, you need to make sure that the fixes get rolled into the new copy of the code if applicable.
- The number of levels allowed in an Application Development Manager project hierarchy. The limit is 25 due to the fact that only 25 libraries can be added to the user portion of the library list for any job. In your situation, the real limit may be less if you need other libraries in the user portion of the library list while doing development. If a new release and its subsequent maintenance hierarchies cause you to approach this limit, you should consider dropping support for your oldest release.
- The effect of supersede chains on your oldest release. As the number of PTFs in a given release of a product grows, the supersede chains for the PTFs can become long. Thus, each new PTF that you create for objects that have already been PTFed at least once results in dragging other objects into your new PTF, including temporary objects and exit programs. This can become difficult to manage after a time. If this happens and the release is old anyway, it is time to consider dropping support for that release.
- Your agreement with your customers as to how long you support an old release. This information should be specified in your maintenance contracts.

10.7.1 Removing Support For an Old Release

This section deals with some of the issues involved in removing an old release from support.

- Notification of customers. You need to give plenty of advance warning to all customers at the old release of your plans to discontinue support for that release. If your releases come on a regular schedule, such as once a year, you can set a policy in your maintenance agreement with your customers as to how many old releases are supported, and at what point in the cycle you discontinue support on an old release.
- Getting your customers onto a new release. You need to provide a means to get your customers to a release which you are still supporting. This can be done by having them apply all PTFs against their old release in order to get their code and files to a level that works with the release to which they are migrating.
- Changes to Application Development Manager structure. Because of the limit to the number of levels in an Application Development Manager hierarchy, as previously described, you want to eliminate the Application Development Manager groups associated with the discontinued release. The groups involved are the development, test, and production groups for the
release, and for the maintenance of the release. The method to clean up these groups is:

1. Ensure that all parts from the groups to be deleted are checked in or promoted to the production levels (for example, PRODUCTION and MAINT_PRODUCTION). To check that no parts are checked out, or exist at a lower level in the hierarchy, you can do a QRYPART:

   `QRYPART PROJ(MLG) GRP(MAINT_DEVELOPMENT) TYPE(*ALL) PART(*ALL) SCAN(*YES)`

   The output from the preceding QRYPART shows all parts in the DEVELOPMENT, TEST, and PRODUCTION groups since the SCAN parameter was set to *YES. The default output of QRYPART is *PRINT shown in the following partial example:

<table>
<thead>
<tr>
<th>Part</th>
<th>Type</th>
<th>Language</th>
<th>Group</th>
<th>Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLGMNUC</td>
<td>CLLESRC</td>
<td>CLLE</td>
<td>PRODUCTION</td>
<td>02/07/96</td>
</tr>
<tr>
<td>MLGMTNC</td>
<td>CLLESRC</td>
<td>CLLE</td>
<td>PRODUCTION</td>
<td>02/07/96</td>
</tr>
<tr>
<td>MLGRPTC</td>
<td>CLLESRC</td>
<td>CLLE</td>
<td>PRODUCTION</td>
<td>02/07/96</td>
</tr>
<tr>
<td>MLGRPTC2</td>
<td>CLLESRC</td>
<td>CLLE</td>
<td>MAINT_DEVELOPMENT</td>
<td>02/08/96</td>
</tr>
<tr>
<td>MLGINQD</td>
<td>DDSSRC</td>
<td>DSPF</td>
<td>PRODUCTION</td>
<td>02/07/96</td>
</tr>
<tr>
<td>MLGMNUD</td>
<td>DDSSRC</td>
<td>DSPF</td>
<td>PRODUCTION</td>
<td>02/07/96</td>
</tr>
<tr>
<td>MLGMSTL</td>
<td>DDSSRC</td>
<td>LF</td>
<td>PRODUCTION</td>
<td>02/07/96</td>
</tr>
</tbody>
</table>

   In the preceding example, you can see that user ABCDEF has part MLGRPTC2 checked out to the MAINT_DEVELOPMENT group. The product administrator needs to discuss the situation with user ABCDEF to find out why the part is checked out, and to either delete it or finish the changes (and generate the subsequent PTF for the changes), and promote the part back to group PRODUCTION.

2. After all parts are at the correct production level, you need to check out all parts that are in the production groups to be deleted, always checking out from the lowest-level group in the hierarchy (which equates to the latest level of the part) into the oldest production group for the release you still intend to support. For example, if you are still supporting Version 2 of the product, but are discontinuing support for Version 1, you should check out all parts that still exist in the maintenance and production groups for Version 1 into PRODUCTION_V2. Make sure your search paths are set up correctly before checking anything out to ensure you see all parts, and see them at the correct level.
3. Create a SCHPTH part in the group to which you checked out all the parts. Edit this SCHPTH part to remove any groups related to the release that you are going to discontinue.

4. Rebuild and retest the application in the group to which you checked out all of the parts. When you build the application, make sure that you have the SCHPTH parameter on the BLDPART set to the name of the search path part you created in the previous step. The application needs to now work exactly the same as the version that was packaged from the group from which you are testing.

5. When you are comfortable that the application was built correctly, you can delete all parts from all groups related to the discontinued release and then delete the groups. You have to delete the groups in the order from lowest to highest in each hierarchy.

**Note**

It is strongly recommended that you save the libraries for any group that you intend to delete prior to the deletion. Keep the save media until the next release is discontinued. You may also want to run the PRTPRJ command before deleting any groups and save the printout with media so you can recreate the project hierarchy if necessary.

### 10.8 Summary

- Concurrent installs of releases without dynamic naming of your libraries makes the upgrade process easier to manage.
- It is advisable to separate the data libraries from your product packaging and provide additional code that can be run at the customer’s convenience to manage their data libraries and the migration of their data to the new release.
- Your new release strategy must also include plans for phasing out support of old releases.
Chapter 11. Supporting Your Product on Previous Releases of OS/400

Since this redbook is aimed at the use of the V3R2, V3R6 and V3R7 release, the previous release to be used is V3R1 or V3R2 or V3R6. Most of the functions we have discussed, such as ILE RPG and ILE CL, were not available at any other previous release. In some areas, it might be helpful to know what other routes to take if you are still supporting your application at a V3R0M5 or a V2R3 release, so we explain the differences for this release. Most of the considerations for support on previous releases is business as usual. However, with the introduction of the RISC processor for the AS/400 system in V3R6, a few things are different from other previous release situations. We assume that the development department finds it unacceptable to maintain two releases of a application with the same functions for two different OS/400 releases.

This chapter describes how to use Application Development Manager to build a version of your product on an AS/400 system at OS/400 V3R6 and install it on a system at a previous release.

We discuss some special considerations that need to be taken into account to save and restore your product in such an environment.

We also describe the process for distributing PTFs from an OS/400 V3R1 to OS/400 V2R3 or V3R0M5, assuming that the products functionally run on such a system (Non-ILE).

11.1 Process Overview

In order to package your product on OS/400 V3R6M0 and install it on OS/400, V3R1M0 you must:

- Create your programs specifying *PRV or V3R6M0 on the Target Release (TGTRLS) parameter of the CRTxxxPGM command.
  
  You must review your programs (including the product’s exit programs) to be sure you are not using a function that is not supported by the previous release of OS/400.

- **DO NOT** remove the observability from your programs.
  
  If you want to move programs between RISC and CISC processors (either way), you need to have the observable code in your programs so the operating system can re-translate the executable code in the format required for that processor.

- Save the product to the target release level.

The Save Licensed Program (SAVLICPGM) command supports the parameter Target Release (TGTRLS).
11.2 Using ADM/400 to Build Your Product With OS/400 *PRV Support

Two solutions are possible for the creation and maintenance of a previous release version of your application in Application Development Manager:

1. Specify the TGTRLS parameter (as V3R1M0, for example) on all BLDOPT parts (on all levels within Application Development Manager):

   This requires a lot of change if this release is no longer valid as a target release and you still want to use the build options. If you only use build options on a project level, this is feasible, but if you have a lot of individual build options for specific programs, this is not recommended.

2. Create a special library with copies of all the Create commands in it that have another default value for the TGTRLS parameter.

   Put that library, for instance, QSYSV3R1, in front of your QSYS library when you are doing development work for the previous release.

   The advantage of this solution is that you can share the same build options for programs in your Application Development Manager hierarchy if you start the creation of a new application release based on functionality in the new OS/400 release. This is the recommended solution.

   This solution is valid for a couple of other commands that have a TGTRLS parameter such as:
   - EXPPART
   - CRTPTF
   - All the SAV.... commands

   Make a copy of all these commands and put them in your special QSYSV3R1 library as well. If all of your development is done for a previous release, you can even change your default system library list, system value QSYSLIBL, with this QSYSV3R1 library permanently ahead of QSYS. The applications and products created for any previous release will always restore on a higher release.

11.3 Saving Your Product to *PRV

On V3R6, V3R7 as well as on V3R2 and V3R1 the SAVLICPGM command supports the Target Release (TGTRLS) parameter, so you can support your application for running as early as V2R3.

One thing that you should consider when supporting your application on both CISC and RISC platforms is the time to translate the application as well as the size of the application. You should maintain multiple copies of the application on systems that have the same OS/400 release that you intend to support. For example, if you are developing your application on a V3R6 system, and want to support it on V3R1 as well, you should install it on both a V3R1 and V3R6 system if those systems are available. By doing this, you can accomplish two things:

- You pay the overhead for the RISC/CISC translation when you restore the product on the V3R1 system. You can then ship this “already translated copy” to your V3R1 customers and their installation does not have to do the translation. You ship to V3R6 customers the copy from your V3R6 machine.
- Since you are shipping an already translated copy of your application, you can remove the observability from both the V3R6 and the V3R1 copies. This can significantly reduce the total size of your application, which can, in turn,
help to reduce media costs or phone charges (if you enable electronic
distribution over switched lines).

11.4 Servicing Your Product with OS/400 *PRV Support.

When you create a PTF using the Create PTF (CRTPTF) command as part of the
PTF generation process, a Save File (SAVF) is created and the PTF objects are
saved into the save file.

Depending on the release were you create this PTF, the following needs to be
done for the creation of a previous release PTF:

1. If your release is V3R6, you specify the target release supported in the
   TGTRLS Parameter (V3R6, V3R1, V3R0M5 and V2R3) on the CRTPTF
   command.

2. If your release is V3R1, the CRTPTF command does not support the Target
   Release parameter.

The first save file created by the Create PTF (CRTPTF) command contains a
PTF for a V3R1 system. The Copy PTF (CPYPTF) command results in a PTF
for a V2R3 system. By renaming the original PTF save file (for V3R1) to a
new name and then copying the PTF (CPYPTF) into the original name, you
now have a PTF for a V2R3 system in a save file by the original name.

11.4.1 V3R1 *PRV Process Step-by-Step

If you want to electronically distribute a PTF created on a system at OS/400
V3R1M0 to a system at OS/400 V2R3M0, you must incorporate the following into
your PTF process:

1. Prior to creating a PTF that is to be distributed to a V2R3 system, ensure all
   program objects for the PTF are created for the target release
   TGTRLS(V2R3M0).

2. Create the PTF using the CRTPTF command:
   
   - CRTPTF PTF('1MP0025') LICPGM(1MLG001) RLS(V1R1M0) +
     LOOID(*CODEDFT) PTFOBJ((MLGINQR230 *PGM)) OBJLIB(MLGPTFLIB) +
     COVER(*NONE) EXITPGM(*NONE)

3. Note the name of the save file the PTF was created into.

   The following messages are found in the joblog:
   
   Library Q1MLG001 created.
   Object MLGINQR230 in Q1MLG001 type *PGM created.
   1 objects duplicated.
   Ownership of object PCCR in Q1MLG001 type *DTAARA changed.
   File Q1MP0025 created in library QGPL
   Object PCCR in Q1MLG001 type *DTAARA deleted.
   Object MLGINQR230 in Q1MLG001 type *PGM deleted.
   Library Q1MLG001 deleted.
   PTF 1MLG001-1MP0025 V1R1M0 created successfully.

4. Rename the PTF save file:
   
   RNMOBJ OBJ(QGPL/Q1MP0025) OBJTYPE(*FILE) NEWOBJ(RNM1MP0025)
   Object Q1MP0025 in QGPL type *FILE renamed RNM1MP0025.

5. Copy the PTF to a save file with the same name as the original PTF save file
   name:
Now you can distribute the previously created and converted PTF to a remote system at OS/400 V2R3M0 using the SNDPTF command:

• Release the PTF from the WRKTP TF display:

```
Work with PTF
System: SYSNAM01
Product ID . . . . . . .: 1MLG001
Release . . . . . . . . : V1R1M0

Type options, press Enter. To work with assigned PTF IDs, press F18.
1=Create 3=Hold 4=Delete 5=Display details 6=Release
9=Work with problems 11=Load/Apply ...

Opt PTF Status PTF save file
  6  1MP0025  Save file only Yes (released)
-  1MP0022  Save file only Yes (held)
-  1MP0020  Save file only Yes (held)
```

• Send the PTF to the Service Requester:

```
SNDPTF PTFID(('1MP0025 1MLG001 V1R1M0'))
PTFs sent to specified service requesters.
```

**Note:** If your PTF distribution process is through tape, you can skip step 4 (RNMOBJ) and copy the PTF to tape from the original PTF save file directly.

### 11.4.2 Observability and Translation Issues With the PTF Process

In Section 11.3, “Saving Your Product to "PRV" on page 154, we talked about how you can maintain two copies of your product on two different levels of OS/400 to avoid the expenses of observability and translation when shipping your products. Unfortunately, there is no real good analogous method for PTFs. Since the PTF objects are shipped in save files, they are not translated until the LODPTF command is run. The LODPTF is part of the process that the customer has to do on the system. You should be aware of this, as it may impact the performance of the LODPTF process, especially on smaller systems.

Probably the best thing that you can do is to stay close to your customers, and develop your product and the PTFs for your product on the OS/400 version that most of your customers are using. This, of course, changes over time, but at least you can minimize the impact of this problem from the perspective of your total customer set. Of course, the fewer PTFs you have, the better (quality pays).

Because of some of the additional Application Development Manager and Application Dictionary Services function, it is probably best to do your application development on a system at OS/400 V3R7, V3R6 or V3R2 if you have that option. However, that does not mean that you have to have your SM/400 license and do your packaging on that same machine. If the majority of your customers are on V3R1, you probably want your SM/400 license on that machine for the reasons stated in the previous two paragraphs. The pictures in Figure 81 on page 157 and Figure 82 on page 157 show how the development and packaging (and PTF creation) can either be done all on one system, or done on two systems at different releases. The only difference is a local versus a remote EXPPART; a remote EXPPART really being a combination of EXPPART/RCVPART.
Notes for Figures

1. This number shows the EXPPART command done prior to packaging.
2. This number shows the packaging program MLGV1PKGCA as discussed in Chapter 7, “Packaging Without the Use of ADM/400” on page 71. You can package your application using this method.
3. This number shows the EXPPART command done prior to CRTPTF.
4. This number shows the CRTPTF process; notice that we have drawn the lines in the diagrams to show that the CRTPTF command process must “touch” the installed product to resolve supersedes, and so on.
11.5 Summary

In this chapter, we discussed some techniques that may help you manage your product across release and CISC/RISC boundaries.
Chapter 12. Additional Packaging and PTF Considerations

This chapter deals with additional considerations regarding packaging of the product and PTFs as well as some authority considerations. It also shows that you can do some of the exit program functions with normal commands instead of using APIs.

12.1 Example of a Minimum Exit Program (PTF) and Its Parameter List

We have already demonstrated that you do not need an exit program for all occasions, but if you need to perform some special actions during the PTF apply/remove process, Figure 83 on page 160 shows you the minimum requirement for that exit program.

Notice that the PTF status and a PTF operation have combinations that never occur.

If you do a permanent apply after loading a PTF, the PTF process runs a Temporary apply first followed by a Permanent apply so your program can be called a maximum of four times (depending on the “Run option” of the EXITPGM parameter. If you specified *PREAPY for instance), then it handles the following situations:

A. 0-5 and 0-1 (temp apply)
and then B. 1-6 and 1-2 (perm apply)

Note

In the preceding example, the x-y numbers correlate to the PTF status and PTF operation parameters as shown in Figure 83 on page 160.
/* Program Name : QPZ1xxxxxx */
/* Note: The PTF Temporary objects (QPZ1xxxxxx) will be deleted */
/* by the PTF process when the PTF is permanently removed or */
/* permanently applied. */
/* PTF temporary objects are not renamed when the PTF is */
/* temporarily removed */
/**/ 02/23/96
 /*------------------------------------------------------------------*/ 02/28/96

/* */ 02/23/96

/* PTF STATUS PTF OPERATION ACTION N=No, Y=Yes, F=Not */
/* ---------- ------------- ------ allowed */
/* 0 0 (REMOVE TEMP) - (Will not occur) */
/* 1 (APP. TEMP) N */
/* 2 (APP. PERM) - (Will not occur) */
/* 3 REMOVE PERM N */
/* 4 (Pre-REMOVE TEMP) - (Will not occur) */
/* 5 (Pre-APP. TEMP) N */
/* 6 (Pre-APP. PERM) - (Will not occur) */
/* 7 (Pre-REMOVE PERM) N */
/* 1 0 (REMOVE TEMP) N */
/* (Applied) 1 (APP. TEMP) - (Will not occur) */
/* 2 (APP. PERM) N */
/* 3 REMOVE PERM N */
/* 4 (Pre-REMOVE TEMP) N */
/* 5 (Pre-APP. TEMP) - (Will not occur) */
/* 6 (Pre-APP. PERM) N */
/* 7 (Pre-REMOVE PERM) - (Will not occur) */
/* 7 (Pre-REMOVE PERM) - (Will not occur) */

PGM PARM(&PRDID &PTFID &RLS &OPTION +
  &PLODID &PTFLIB &USRDTA &PTFSTS +
  &PTFOPER)
 /*------------------------------------------------------------------*/ 02/28/96

MONMSG MSGID(CPF0000) EXEC(GOTO CMDLBL(DAMAGED)) 02/23/96
 /*------------------------------------------------------------------*/ 02/28/96

DAMAGED: 02/23/96
QSYS/SNDPGMMSG MSGID(CPF3639) MSGF(QCPFMSG) MSGTYPE(*ESCAPE) 02/23/96
 /*------------------------------------------------------------------*/ 02/23/96
ENDPGM 02/23/96

* * * * E N D O F S O U R C E * * * *

Figure 83. Example of an Exit Program Framework
12.2 Example of a Product Exit Program and Its Parameters

A product exit program template is found in SG244187.1/QSRCCH11. This template can be used for exit programs that are called by the RSTLICPGM, SAVLICPGM, DLTLICPGM, and CHKPRDOPT commands for either *CODE or *LNG loads.

You should note that the program can be called at the times shown in the following table.

<table>
<thead>
<tr>
<th>*BEFORE</th>
<th>*AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSTLICPGM</td>
<td>RSTLICPGM</td>
</tr>
<tr>
<td>SAVLICPGM</td>
<td>SAVLICPGM</td>
</tr>
<tr>
<td>DLTLICPGM</td>
<td>-----</td>
</tr>
<tr>
<td>----</td>
<td>CHKPRDOPT</td>
</tr>
</tbody>
</table>

Since there is no post-delete exit program, you can run into a problem. Suppose that your application objects all need to be owned by user ID XYZ. At pre-RSTLICPGM time, your exit creates the user profile XYZ and all objects in the product are restored to this user ID. Now you want to delete the product. However, if you delete user profile XYZ in your pre-DLTLICPGM exit, the user profile is not deleted because the default for the OWNOBJOPT parameter on the DLTUSRPRF is *NODLT, which means “do not delete the profile if it owns any objects”. You should code the DLTUSRPRF command in your exit program to set the OWNOBJOPT parameter to *CHGOWN instead, and specify a new owner such as QSYS. The reason you want to use *CHGOWN and not *DLT on the OWNOBJOPT parameter is because *DLT results in the deletion of most (or all) of your product, which should be left up to the DLTLICPGM code. The price you pay is that the DLTLICPGM takes longer because the ownership of all of the objects has to be done before the process can proceed.
/* PGM Name PRDEXITXMP */ 02/28/96

/* According to the SM/400 use manual, the valid combinations */ 02/28/96

/* for the function and indicator parameters are: */ 02/28/96

/* FUNCTION INDICATOR */ 02/28/96

/* *RSTCODE/*RSTLNG *BEFORE or *AFTER */ 02/28/96

/* *SAVCODE/*SAVLNG *BEFORE or *AFTER */ 02/28/96

/* *DLTCODE/*DLTLNG *BEFORE only */ 02/28/96

/* *CHKCODE/*CHKLNG *AFTER only */ 02/28/96

/* */ 02/28/96

/* This program will return message CPF3D95 'Exit program */ 02/28/96

/* processing failed */ 02/28/96

/* to the calling program for any unmonitored */ 02/28/96

/* error condition. */ 02/28/96

/* */ 02/28/96

/********************************************************************/ 02/28/96

PGM PARM(&FUNCTION &INDICATOR &LNGID &EXITPGMLIB + 11/19/93
 &CURRENTLIB &ACREATELIB &ARSTLIB &ACURRENTFLR + 11/19/93
 &ACREATEFLR &ARSTFLR &ACURVRM &ARSTVRM + 11/19/93
 &LMISSOBJ &NRMMISSOBJ &LMISSFLR &NRMISSFLR) 11/19/93

02/28/96

EXIT pgm PARMS */ 11/19/93

DCL VAR(&FUNCTION) TYPE(*CHAR) LEN(10) 11/19/93

/* Function being performed - valid values are: */ 02/28/96

/* *RSTCODE */ 02/28/96

/* *SAVCODE */ 02/28/96

/* *DLTCODE */ 02/28/96

/* *CHKCODE */ 02/28/96

/* *RSTLNG */ 02/28/96

/* *SAVLNG */ 02/28/96

/* *DLTLNG */ 02/28/96

/* *CHKLNG */ 02/28/96

DCL VAR(&INDICATOR) TYPE(*CHAR) LEN(10) 02/28/96

/* *BEFORE or *AFTER indicator */ 02/28/96

DCL VAR(&LNGID) TYPE(*CHAR) LEN(4) 11/19/93

/* Language id */ 02/28/96

DCL VAR(&EXITPGMLIB) TYPE(*CHAR) LEN(10) 11/19/93

/* Library that contains exit programs */ 02/28/96

DCL VAR(&CURRENTLIB) TYPE(*CHAR) LEN(10) 11/19/93

/* Library where product currently exists */ 02/28/96

DCL VAR(&CREATELIB) TYPE(*CHAR) LEN(10) 11/19/93

/* Library specified by creator of product */ 02/28/96

DCL VAR(&RSTLIB) TYPE(*CHAR) LEN(10) 11/19/93

/* Library where product is going to exist */ 02/28/96

DCL VAR(&CURRENTFLR) TYPE(*CHAR) LEN(12) 11/19/93

/* Root folder that product currently uses */ 02/28/96

DCL VAR(&CREATEFLR) TYPE(*CHAR) LEN(12) 11/19/93

/* Root folder specified by creator */ 02/28/96

DCL VAR(&RSTFLR) TYPE(*CHAR) LEN(12) 11/19/93

/* Root folder that product will use */ 02/28/96

DCL VAR(&CURVRM) TYPE(*CHAR) LEN(6) 11/19/93

/* Release level of the product */ 02/28/96

DCL VAR(&RSTVRM) TYPE(*CHAR) LEN(6) 11/19/93

/* Release level of the product being restored */ 02/28/96

DCL VAR(&LMISSOBJ) TYPE(*CHAR) LEN(9980) 11/19/93

/* Array of missing objects and their types */ 02/28/96

DCL VAR(&NRMISSOBJ) TYPE(*CHAR) LEN(4) 11/19/93

/* Number of missing objects */ 02/28/96

DCL VAR(&LMISSFLR) TYPE(*CHAR) LEN(6300) 11/19/93

/* Array of missing folders */ 02/28/96

DCL VAR(&NRMISSFLR) TYPE(*CHAR) LEN(4) 11/19/93

/* Number of missing folders */ 02/28/96

MONMSG MSGID(CPF0000) EXEC(GOTO CMDLBL(STDERR)) 02/28/96

Figure 84 (Part 1 of 2). Example of a Pre-Operation and Post-Operation Program Framework
Figure 84 (Part 2 of 2). Example of a Pre-Operation and Post-Operation Program Framework
12.3 Authorization Considerations for Packaging and Exit Programs

In Section 8.10, “PTF Process Tips and Hints” on page 102, information regarding authority considerations during the packaging process is already discussed.

Another point of interest is the authority that a pre-operation or post-operation product exit program runs under. What happens during the RSTLICPGM process:

- RSTLICPGM runs under control of QSYS user profile.
- At the time that RSTLICPGM gives control to the pre-operation or post-operation program, it revokes all propagated authorities. So the rights that your PSTOPR program runs under are:
  - The rights of the profile that uses the RSTLICPGM,
  - The rights of the owner of the PSTOPR program assuming it is created with USRPRF(*OWNER).

So, if your exit program needs to create user profiles or to do other actions that must be done by a user profile with *SECADM authority, make sure your install program is owned by QSECOFR and *OWNER before the SAVLICPGM. This gives all of the control over the process to the pre-operation or post-operation program. Otherwise, you must specify in your install instructions that the install is to be performed by QSECOFR or an exact equivalent of QSECOFR in terms of authorities.

12.4 To Use or Not to Use - APIs

In most of the examples in this redbook, we show the use of APIs in exit programs since they give the best performance. However, if you are not comfortable in programming with APIs, you can still use all of the ordinary CL commands that you are familiar with such as:

- **DSPOBJD**
  Instead of using an API to store all of the information regarding objects from a library in a user space, you can do a DSPOBJD to an outfile and read the outfile.

- **CHGPRDOBJD**
  Updating the OIR can be done by an API but the command CHGPRDOBJD performs the same task. There is one minor difference; when you use the API, you can reset the product information in the OIR to blanks if you inadvertently marked an object incorrectly. CHGPRDOBJD does not allow you to input blank character strings.

There is no companion CL command for the API that retrieves the information about the product (QSZRTVPR). Its use is demonstrated in several exit programs.

There are APIs for:

- Create Product Definition (QSZCRTCPRD)
- Create Product Load (QSZCRTPL)
• Package Production Option (QSZPKGPO)
• Other packaging functions

For the complete list, see the System API Reference. If you are proficient in coding with APIs, you could conceivably create your own custom packaging commands or programs.

12.5 Why Do We PTF ILE Programs and Not Modules?

Now that we have been working with the new ILE application structure using modules as small entities, you might also know that you can update a program with modules. Application Development Manager does not support this kind of update and always re-binds the complete program.

On first sight, you get the impression that shipping modules in the PTF process saves on the size of the PTF to be shipped electronically. This is true in most cases but the savings are not as much as you might think, as we see later.

You pay the price of shipping modules in another form; let’s discuss some of the disadvantages:

1. An Exit program is required:

   Since the PTF process only deals with the replacement of objects (of course, modules are also objects), updating the programs in which the modules reside needs to be done by an action in your exit program such as UPDPGM or UPDSRVPGM. Imagine that you also want to support the removal of a PTF, then your exit program needs the old module at removal time to replace the new module again with the UPDPGM command. There are two ways to get that old module:

   • Have all of your modules packaged with the product so the PTF process replaces the old object (module) and keeps it around until the PTF is permanently applied.
   • Send the old version of the module in the PTF together with the new version and save it until the PTF is permanently applied.

   You can see that the exit program design becomes quite difficult.

2. You need to administer the supersedes:

   In the PTF environment, the supersedes are recognized by the PTF process. This happens based on the objects that are shipped in a PTF and are compared with all of the objects that have ever been shipped previously in another PTF. A recognition makes the previous PTF a prerequisite for this PTF and, therefore, includes all of the superseded PTF objects in this PTF. Take a look at the following scenario in Figure 85 on page 166.
Assume that we do not have all of the modules in our product packaged; here is what happens in this scenario.

PTF-1 Actions

a. We package the OLD and NEW module-A in the PTF. The new module is called module-A.

b. Code the actions for update and removal in our exit program.

c. Check for supersedes.

d. PTF size: Two modules and one exit program.

PTF-2 Actions

a. We package the OLD and NEW module-B in the PTF.

b. Code the actions for update and removal of module-B in our exit program.

c. Manual of programmed check for supersedes; we require that PTF-1 is installed since we have never tested a situation with the new module-B and the old module-A.

d. We package the OLD and NEW module-A in PTF-2.

e. Code the actions for update and removal of module-A in our exit program.

f. PTF size: Four modules and one exit program.

3. Manage your own OIR update for the programs:

In the PTF process, the OIR of the objects packaged in the PTF is updated. Since we are responsible for the updating the program in the product, it is our responsibility in the exit program to do the OIR updating after we have updated that program (if we want to track the PTF level at the program versus module level).

What should have happened if we had used only the program for the PTF process:

- For PTF-1, Package PGM01 for the PTF, no exit program.
- For PTF-2, Package PGM01 for the PTF, the PTF process recognizes that PTF-1 is superseded and adds all of the objects to this PTF, but since this is the same program, nothing happens. Once again, no exit program.
- In both cases, the size of the PTF is only the size of this PGM01 program.

We think you agree that leaving the PTF process in charge is the most effective way of handling PTFs.
12.6 What Is the Role of QPZ1 Prefixed Objects

As you have seen in earlier chapters, PTF objects or exit programs that begin with "QPZ1" are considered temporary objects by the PTF process. The temporary objects serve two purposes:

1. At the end of the permanent apply process, they are deleted automatically. Any additional objects that are needed by your PTF exit program can be named as temporary objects so that your product does not get "polluted" with extra objects.

2. While loading a PTF, all of the objects involved in the PTF process are renamed and loaded in the product library. So at pre-apply time, these renamed objects are not available for any action by your exit program because you have no control over the naming scheme used. The only objects available at that time are the "QPZ1" prefixed objects.
Chapter 13. Advanced topics

This chapter discusses some topics that are useful if you are not familiar with the subjects. However, in order to understand the subject of the redbook, it is not necessary to study all of this. The following subjects are discussed:

- 13.1.3, “ADM/400 and ADS/400” on page 172
- 13.5, “License Management Overview” on page 181
- 13.6, “Using SystemView System Manager/400 to Manage Software Changes” on page 193
- 13.7, “First Failure Data Capture (FFDC)” on page 212

13.1 Tips Using Application Development Manager for the ILE Application Development

If you are not familiar with the Redbook, *Moving to Integrated Language Environment for RPG IV*, GG24-4358, the next two paragraphs give you some more information on the relations between Application Development Manager and ILE.

13.1.1 Naming Conventions

Since there are so many objects (parts) involved in the development process of ILE applications, some naming conventions are very helpful to identify relations and dependencies between the different parts. Let us first start with a list of typical part types involved:

- Source
- Module
- Program
- Service program
- Binding source
- Binding directory
- Build option

The build option (BLDOPT) is a very important part since it can influence the creation of other parts. Application Development Manager takes, by default, the normal create commands from the system. The next level of control occurs when you create a QDFT BLDOPT part in which you describe your defaults for a certain project or group. And for complete control, you can make a BLDOPT part with the same name as the source part that you want to build (create). This overrules all of the other BLDOPT create definitions.

Creating a PGM requires a build option part with the same name as the entry point module.

Figure 86 on page 170 shows an example of how this is used for documentation as well.
Figure 86. Example of a Build Option for Documentation

Figure 86 shows you what program is built, which modules were used, what service programs are bound, and the activation group it is run in. Remember that without the use of Application Development Manager, you have to document this somewhere else; otherwise, you are losing control in the change management process.

If you are not using Application Development Manager and you need to re-create the program, all information needed for a re-create is available in the program description itself.

Note: The UPDPGM and UPDSRVPGM commands are not supported in Application Development Manager.

13.1.2 Relationships

What are the relations between some of the parts that are mentioned before?

<table>
<thead>
<tr>
<th>Type</th>
<th>Creates</th>
<th>Has a create-relation with</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE</td>
<td>MODULE</td>
<td></td>
</tr>
<tr>
<td>MODULE</td>
<td>PGM</td>
<td>Binding directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entry module (PEP)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Service program</td>
</tr>
<tr>
<td>MODULE</td>
<td>SRVPGM</td>
<td>Binding source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Binding directory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Service program</td>
</tr>
</tbody>
</table>

Figure 87. Create Relations

In Application Development Manager, the same table should be constructed as follows:
Create Part Triggered by Has a create-relation with
==================================================================
MODULE SOURCE
==================================================================
PGM MODULE Binding directory
Other modules
Entry module (PEP) (E-module)
Service program
SRVPGM BNDSRC Binding source
Binding directory
Other modules
Service program

Figure 88. Trigger Relations

So a naming convention that fits the Application Development Manager environment can be similar to the following:

<table>
<thead>
<tr>
<th>Create Part</th>
<th>Involved parts</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE = modulename</td>
<td>SOURCE = modulename</td>
<td>Required</td>
</tr>
<tr>
<td>BLDOPT = QDFT</td>
<td></td>
<td>Optional Project default</td>
</tr>
<tr>
<td>PGM = E-modulename</td>
<td>MODULE = E-modulename</td>
<td>Required</td>
</tr>
<tr>
<td>MODULE = modulename</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>SRVPGM = servicepgm</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>BNDDIR = binddirname</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>BLDOPT = E-modulename</td>
<td>Contains the CRTPGM with</td>
<td></td>
</tr>
</tbody>
</table>
| | MODULE | .....
| | TEXT | .....
| | ENMOD | .....
| | BNDSRVPGM | .....
| | BNDDIR | .....
| | ACTGRP | .....
| SRVPGM = servicepgm | MODULE = E-modulename | Required |
| BNDSRC = servicepgm | Required |
| BNDDIR = binddirname | Optional |
| MODULE = modulename | Optional |
| BLDOPT = servicepgm | Required |

Figure 89. Part Naming Rules

Using this kind of naming convention is probably not taking care of everything, for example, if you are using the E-module (Entry module) also as a non-entry module in other program binds, the naming convention is broken. But you need to think about using some kind of a naming convention in your environment since there are so many more objects (parts) involved today than in an OPM environment.

Note: The default build commands for RPGLESRC and CBLLESRC are CRTBNDRPG and CRTBNDCBL. In order to build modules from these source types, you must have a BLDOPT.
13.1.3 ADM/400 and ADS/400

Figure 90 illustrates the relationships between:

- The AS/400 system, the Operating System/400, and the different features of Application Development ToolSet/400
- Application Development Manager, Application Dictionary Services, the features of Application Development ToolSet Client Server/400
- CODE/400, VRPG Client/2, and SM/400

![Diagram of Application System/400 Environment]

From CODE/400 and Application Dictionary Services, developers also have access to the functions of the Application Development Manager product.

13.2 ADS/400

The Application Dictionary Services programming tool integrates a dictionary database program with AS/400 application development tools. It stores descriptions of application objects and their relationships in an application dictionary, thereby maintaining a complete inventory of all objects. As you make changes to these objects, the information in the dictionary is updated. For detailed information on the Application Dictionary Services product, see the Application Dictionary Services User’s Guide.
13.2.1 Overview of the ADS/400 Functions

The ADS/400 tool can help you design and maintain your applications in the following ways:

- ADS/400 V3R2, V3R6 and V3R7 supports ILE binding references.
- To determine where fields are used in the objects that constitute your application and to determine what files, programs, or other objects are affected by a change to a given field.
- To ensure that a change to a field is reflected in all objects that refer to that field and, if necessary, to recompile files, programs, or other objects that are affected by this change.
- To identify certain physical files as field reference files.
- To create physical files based on the field reference files.
- To search for strings in a source member or to scan for externally described field names in RPG programs to determine if change to that field has real impact.
- To determine which programs are calling a specific program or which programs are being called by a specific program.
- To build SQL table files based on the field reference files and build SQL index files based on the SQL table files and physical files.

The Application Dictionary Services tool has an interface to Application Development Manager so you are able to use Application Development Manager options on Application Dictionary Services displays. In a standard Application Dictionary Services session you document information about objects that reside in libraries. In a combined Application Dictionary Services Application Development Manager session, you do this for objects that reside in groups. All of the functions described previously are available in a combined session, except the last function.

13.3 Our MLG Project Dictionary

The following steps are needed to create a dictionary:

1. Create a new library with the CRTLIB command:

   CRTLIB LIB(ADSMLGDT) TYPE(*PROD) TEXT(‘ADS/400 dictionary for our MLG Project’)

2. Enter the STRADS command followed by the parameters described in the following steps:

   a. The name of the application dictionary to be created (ADSMLGDT).
   b. Whether this is an Application Development Manager session (*YES).
   c. The name of the project from which groups are documented (MLG):

      STRADS APPDCT(ADSMLGDT) ADM(*YES) PRJ(MLG) GRP(DEVELOPMENT)

3. The “Work with Documented Groups” display appears.
4. Press F6 to get a list of all of the groups in the MLG project.

5. After selecting all of the groups, press Enter to submit the dictionary creation job to the batch system.

13.4 Working with the MLZIP Field

Once we have created our dictionary, we can use it to get detailed information about specific fields and about programs and files that are affected by a change to a field. This chapter shows you how to:

- Determine the MLZIP field reference hierarchy.
- Determine which files and programs are affected by the change to the MLZIP field in the following sequence:

  1. Field Reference File
  2. Physical Files
  3. Logical Files
  4. Display Files
  5. Programs.
  6. Service programs
  7. Modules
13.4.1 The MLZIP Field Reference Hierarchy

Start the Application Dictionary Services session with the command:

```
STRADS APPDCT(ADSMMLGDT) ADM(*YES) PRJ(MLG) GRP(MAINT_DEVELOPMENT)
```

1. From the Application Dictionary Services menu, type 1.
2. Press Enter. The “Subset Fields to Work With” display appears.

```
Subset Fields to Work With
 Dictionary ................. : ADSMLGDT
 Project ................. : MLG

 Field ....................... MLZIP *ALL, name, generic*
 Record ..................... *ALL *ALL, name, generic*
 Part ....................... *ALL *ALL, name, generic*
 Attribute .................. *ALL *ALL, attribute

 Text:
 Search words .............
 Search condition ........ 1 1=Or, 2=And

 F4 for list
```

```
Figure 93. Subset Fields to Work With Display
```

3. Fill in the fields and press Enter. The “Work with Fields” display appears listing the fields we requested.

```
Work with Fields
 Dictionary ........ : ADSMLGDT

 Field Record Part Group
 MLZIP DETAIL MLGNAMD PRODUCTION
 MLZIP DSPLY2 MLGINQD PRODUCTION
 MLZIP DSPLY2 MLGTMND PRODUCTION
 MLZIP MLGSTR MLGMSLT PRODUCTION
 MLZIP MLGSTR MLGMSLT2 PRODUCTION
 MLZIP MLGSTR MLGMSLT3 PRODUCTION
 MLZIP MLGSTR MLGMSTP PRODUCTION
 MLZIP MLGSTR MLGNAML PRODUCTION
 MLZIP MLGREFR MLGREFP PRODUCTION
 MLZIP SFLRCD MLGNAMD PRODUCTION

 Command

 F3=Exit F4=Prompt F5=Refresh F9=Retrieve
 F12=Cancel
```

```
Figure 94. Work with Fields Display
```

Note: In an Application Development Manager session, Application Dictionary Services searches for the parts using the search path specified on the “Start AppDict Services/400” display. Only the first occurrence of each field in the search path is listed on the display.
Use F21=Print List to print a report of the Field List. The list is shown in Figure 98 on page 178 and is used later.

4. Type 8 beside the MLZIP file MLGREFP.

5. Press Enter. The “Work with Impacted Files” display appears listing all of the parts documented in the dictionary that are affected by a change to this field.

Note: In an Application Development Manager session, if the impacted parts are not documented in the dictionary, they are not listed. However, they are listed if the impacted parts are documented in the dictionary and not found in the specified search path.

Use F21=Print List to print a report of the impacted files. The list is shown in Figure 99 on page 179 and is used later.

The display shows all of the files that are impacted by the change of the MLZIP field in the field reference file. If we want to see the impact of the change, we need to look at the source of each individual part. There is no “find string” option on these displays. We need to use option 2 to look at the source. Application Dictionary Services does a checkout to the group specified in the STRADS command (MAINT DEVELOPMENT in the current situation).

This is a disadvantage when you use Application Dictionary Services and Application Development Manager together. Normally, you want to do an impact analysis without doing an implicit checkout to another group. With this Application Dictionary Services approach, we need to delete the source parts that require no change and are checked out by Application Dictionary Services after the impact analysis.
Circumvention for CHECKOUT

If you have implemented the "DS" user option as mentioned in Section B.6, "Browsing a Source Part" on page 264, you can browse the source related part of an object without checking it out.

6. From the "Work with Impacted Files" display, press F15. The "Work with Impacted Programs and Modules" display appears listing all of the programs that refer to all of the files listed on the previous display.

**Note:** In an Application Development Manager session, if the impacted parts are not documented in the dictionary, they are not listed. However, they are listed if the impacted file parts are documented in the dictionary but are not found in the specified search path.

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>ADSMLGDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>MLZIP</td>
</tr>
<tr>
<td>Group</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>Project</td>
<td>MLG</td>
</tr>
</tbody>
</table>

**Work with Impacted Programs and Modules**

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>ADSMLGDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>MLZIP</td>
</tr>
<tr>
<td>Group</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>Project</td>
<td>MLG</td>
</tr>
</tbody>
</table>

Type options, press Enter.

2=Edit 4=Delete 14=Build 15=Scan RPG source
25=Find string 28=Check out 29=Check in 30=Promote

<table>
<thead>
<tr>
<th>Opt</th>
<th>Part</th>
<th>Type</th>
<th>Attribute</th>
<th>Use</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>MLGINQR</td>
<td>MODULE</td>
<td>RPGLE</td>
<td>03</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGLBLR</td>
<td>MODULE</td>
<td>RPGLE</td>
<td>01</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGMNTR</td>
<td>MODULE</td>
<td>RPGLE</td>
<td>03</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGMNMR</td>
<td>MODULE</td>
<td>RPGLE</td>
<td>07</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGRPTC</td>
<td>MODULE</td>
<td>CLLE</td>
<td>08</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGRPTC2</td>
<td>MODULE</td>
<td>CLLE</td>
<td>08</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGRPTR</td>
<td>MODULE</td>
<td>CLLE</td>
<td>01</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGRPTC2</td>
<td>MODULE</td>
<td>PGM</td>
<td>07</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGRPTC</td>
<td>MODULE</td>
<td>PGM</td>
<td>01</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGRPTC2</td>
<td>MODULE</td>
<td>PGM</td>
<td>01</td>
<td>PRODUCTION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Opt</th>
<th>Part</th>
<th>Type</th>
<th>Attribute</th>
<th>Use</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>MLGINQR</td>
<td>MODULE</td>
<td>RPGLE</td>
<td>03</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGMNTR</td>
<td>MODULE</td>
<td>PGM</td>
<td>07</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGRPTC</td>
<td>MODULE</td>
<td>CLLE</td>
<td>01</td>
<td>PRODUCTION</td>
</tr>
<tr>
<td>25</td>
<td>MLGRPTC2</td>
<td>MODULE</td>
<td>PGM</td>
<td>01</td>
<td>PRODUCTION</td>
</tr>
</tbody>
</table>

F3=Exit  F4=Prompt  F5=Refresh  F6=Recreate parts
F10=Command entry  F12=Cancel  F24=More keys

*Figure 96. Work with Impacted Programs Display*

Use F21=Print List to print a report of the impacted programs and modules. The list is shown in Figure 100 on page 179 and is used later.

7. Use option 25, "Find string," against the first program/module in the list. To find the string in all of the programs/modules listed (if the module or program is source related, it depends on the type OPM or ILE), press F13 to repeat the action until you reach the end of the listed programs/modules. Press Enter and Application Dictionary Services shows you the "Find String Using PDM" display.

**Note:** When you use option 15, "Scan RPG source," on the "Work with Impacted Programs" display, Application Dictionary Services submits a batch job and gives you a report. Obviously, you cannot use this option for CLP parts.
8. If you use option *NONE 1, you might as well run this action in batch since you eliminated the interactive capabilities. The print list option 2 gives you a list of sources that has a reference to the MLZIP field in the source. In the output produced from this action, you find a spooled file for every source examined. The only spooled file that you should look for is a spooled file with the name: QDNUPFST. An example of this report is shown in Figure 101 on page 179 and contains only the members that have a reference to the field.

13.4.2 Result of the First Stage of the Impact Analysis

So far we have produced the following reports that are used as a base for our further analysis process.
13.4.3 Second Stage of the Impact Analysis Process

Depending on your application design, you can start with the report shown in Figure 98 or the report shown in Figure 99. Based on our knowledge that all fields are referenced from the Field Reference File, we start with the second report.

Follow the next steps to complete the second stage:
1. Return to the “Work with impacted files” display and use the “DS” user option to inspect the sources.
   - The MLGREFP file needs a change for the MLZIP field length.
   - The MLGMTND display file has a message related to the MLZIP field that needs to be changed as well. To be honest, you need to know your application to catch relations such as this, or you should have it documented in some other place. This change implies, at the same time, that a change needs to be made in the message file MLGMSGF. Since MSG00086 is also used for another purpose, the message ID has to be changed; we chose the new number MSG0901.
   - Document all information (findings) on your report, which finally should look similar to the following:

   ** Written Instructions **

   Change MLGREFP, MLZIP field to length 6.0
   Change MLGMTND, line 108 message-id from MSG00086 to MSG0901
   Add message-id MSG0901 to message file MLGMSGF text="Numeric 6.0"

   Figure 102. Impacted File List

2. Take the report from Figure 101 and get to the “Work with Impacted Programs and Modules” display. Analyze the three modules.
   - MLGLBLR and MLGMTNR do not need a change.
   - MLGRPTR needs to be changed to keep the MLZIP data correctly under the field header.

3. Document all of the information on the report, which looks similar to the following:

   ** Written Instructions **

   Change MLGRPTR, line 39 to output position 72

   Figure 103. Source Members Containing the MLZIP Field

4. The result of the impact analysis is summarized in Table 3 on page 181.
Table 3. Impact Analysis for the MLG Application

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Part Type</th>
<th>Source Change</th>
<th>Source Change Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLGREFP</td>
<td>DDSSRC</td>
<td>Yes</td>
<td>Field MLZIP to length 6.0</td>
</tr>
<tr>
<td>MLGMTND</td>
<td>DDSSRC</td>
<td>Yes</td>
<td>Line 108, change MSGID MSG0086 to MSG0901</td>
</tr>
<tr>
<td>MLGMSGF</td>
<td>MSGF</td>
<td>Yes</td>
<td>Add MSGID MSG0901, “Numeric 6.0”</td>
</tr>
<tr>
<td>MLGRPTR</td>
<td>RPGLESRC</td>
<td>Yes</td>
<td>Line 39, change pos 71 to 72</td>
</tr>
</tbody>
</table>

13.4.4 After the Impact Analysis

After the impact analysis is completed, two possible options are available to perform the real changes. Each option has advantages and disadvantages:

1. Do all of the changes within Application Dictionary Services:
   - **Advantage**: Because Application Dictionary Services includes the options “Build”, “Check out”, “Check in”, and “Promote”, you do not need the Application Development Manager user interface.
   - **Disadvantages**: The concept of Application Dictionary Services is totally different from Application Development Manager.
     - The PARTS Application Dictionary Services shows you are objects, not sources.
     - The group where these object parts reside is shown in the “Group” field on the Application Dictionary Services panel. This group need not be the same as where the source part resides. This may be confusing.

2. Use Application Development Manager to do the changes:
   - **Advantages**: User interface integrated with PDM.
     - Very clear interaction when you work with parts.
   - **Disadvantage**: Need to switch between Application Dictionary Services and Application Development Manager sessions.

13.5 License Management Overview

OS/400 includes APIs to allow you to implement license management of your Licensed Program Products. License management is the monitoring of user compliance with the license agreement for a software product.

The type of **compliance** determines the action taken by the product when the authorized limit is reached. OS/400 supports three types of compliance:

- **Warning**: A user is never denied access to the product. When the usage limit is exceeded, a message is sent to the product and QSYSOPR indicating that the limit has been exceeded.
- **Operator Action**: When the usage limit is reached, new users are denied access to the product. The new user cannot access the product until the usage limit is increased or a license user is released.
• **Keyed** - A software license key is required from the vendor to install or to increase the usage limit of the software. The software provider may permit a grace period that allows the usage limit of a product to be exceeded by 50 percent for "n" days before a new software license key is required to allow additional users access to the product. If a new software license key with an increased usage limit is not installed prior to the end of the grace period, the usage limit of the product reverts back to the entitled usage limit.

OS/400 License Management supports two types of license usage:

• **Concurrent** - This is the number of unique jobs that can be using the product at the same time.

• **Registered** - This is the number of unique license users registered to use the product.

**Using License Management:** For the license management function to monitor usage of your product, you must package your product. As part of the packaging process, you define license usage information with the Add Product License Information (ADDPRDLICI) command. The license usage information is packaged and shipped with your product and includes the usage type, compliance type, license term, and default usage limit.

Your product requests use of the license through a call to the Request License API, QLZAREQ. Typically, this request is in a central program that all users of the product go through. The license request causes the usage count to be incremented. To release the license, the product places a call to the Release License API, QLZARLS, specifying the same parameters as on the license request. This causes the license usage count to be decremented.

Detailed information on License Management is in Chapter 4, “Packaging User-Base Priced Products, of System Manager/400 Use.”

**API Authority Considerations:** At V3R2 and later releases, all of the license management APIs are shipped with PUBLIC(*USE) authority. Prior to V3R2, all of the license management APIs were shipped with PUBLIC(*EXCLUDE) authority. For the product to run on a system prior to OS/400 V3R2, your product should ship a user profile that has private authority to the APIs. This ensures that the request and release of licenses is successful regardless of the authority of the user running the product. This is accomplished by:

• Compiling the program that calls the APIs with the user profile parameter set as *OWNER.

• Using an exit program to create a user profile, grant object authority for the APIs to the profile, and change object ownership of the program calling the APIs to the user profile.

• Specify the exit program in the post-operation exit program parameter of the create product load command.

Following these steps ensures that the program calling the license management APIs runs under a user profile that has authority to the APIs.
To enable your application for License Management, your product must be packaged as described in Section 6.2, “Converting Your Application Into a Packaged Product” on page 52. For License Management, the concept of feature is introduced. A feature is a separately-priced piece of a product that may contain one option or a number of options. All of the options with the same code load ID belong to the same feature.

The product license information must be added when the product is packaged. License information is added to your product with the Add Product License Information (ADDPRDLICI) command:

```
ADDPRDLICI PRDID(1MLG002) RLS(V2R1M0) +
  USGTYPE(*CONCURRENT) COMPLIANCE(*KEYED) +
  DFTUSGLMT(2) LICTRM(*VERSION) VNDPWD(*NONE) +
  GRACE(30 *NO)
```

Some of the fields in the ADDPRDLICI command are:

- **USGTYPE** - You specify which type of access your product uses, *CONCURRENT or *REGISTERED.

- **COMPLIANCE** - You specify which type of compliance scheme you want to set up for your product. The choices are *KEYED, *OPRACTION, and *WARNING.

- **LICTRM** - This is the level of the product or feature for which the authorized usage limit is valid. The choices are *VERSION, *RELEASE, or *MOD. You probably want to set this to coincide with your new release strategy. For example, if you always package new versions of your product instead of new releases, then you want to set this parameter to *VERSION. This may imply that you should decide on your future release strategy when you first package a product for License Management.

- **DFTUSGLMT** - This is the default usage limit that is shipped with the product and is used until a license key is entered that changes the usage limit. You normally want to set this value to 0, or some small number. If you are sending your application to customers on a trial basis, you need to decide whether you want to allow one or two users on an indefinite basis (DFTUSGLMT = 1 or 2), or whether you want to set the DFTUSGLMT to 0 and generate a temporary license key that expires at the end of the trial period.

  You should also be aware that the number of users specified in the DFTUSGLMT parameter is the number of users that are allowed to continue to use your product after the license expires.

- **GRACE** - this parameter contains two values:
  - Number of days: the number of days that a product can be utilized (by users over the usage limit) once the usage limit has been exceeded. Usage may only exceed the usage limit by 50% during the grace period. For example, if the grace period is 30 days, the usage limit is 20. When the 21st user accesses the product on August 5th, the user has until September 3 to get a new license key to increase the usage limit. After September 3, any license requests cause any users over the usage limit
to be released. Once these users exit the product, they are not allowed to access the product or feature again until a new license key is installed with additional usage, or the number of users becomes less than the authorized usage limit.

- Allow for default usage limit: this field determines whether or not the default usage limit can be exceeded prior to the installation of a license key. Setting this field to NO means users are not allowed to exceed the default usage limit before the software license key is installed. Setting this field to YES means users are allowed to exceed the default usage limit for the number of days in the grace period before the software license key is installed.

- VNDPWD - this is a password that is encrypted and stored with the product license information. It is the same password that is used to generate software keys for this product. This ensures that software license keys for your product are only generated by the person (or persons) who know the password. The password must begin with an alphabetic character (A through Z, $# or @) and be followed by 9 or less alphanumeric characters (A through Z, $,#,@ or 0 through 9).

### 13.5.2 Enabling an Application Under Keyed License Management

To enable an application for use under keyed license management, you need to obtain a license key from the software provider who runs the GENLICKEY command. After you obtain the license key, you can run the ADDLICKEY command, which enables the application based on the terms of the license key.

#### 13.5.2.1 The GENLICKEY Command

This command is run by the software provider when a customer requests the purchase of a license. The resulting key is a combination of 18 characters and numbers, A-F and 0-9. The product definition of the product for which you are generating the key must exist on the system when the GENLICKEY command is run.

---

**Generate License Key (GENLICKEY)**

Type choices, press Enter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product ID</td>
<td>1MLG002</td>
</tr>
<tr>
<td>License term</td>
<td>V2, Vx, VxRy, VxRyMz</td>
</tr>
<tr>
<td>Feature</td>
<td>5001</td>
</tr>
<tr>
<td>System serial number</td>
<td>1234567</td>
</tr>
<tr>
<td>Processor Group</td>
<td>*ANY</td>
</tr>
<tr>
<td>Expiration date</td>
<td>*NONE Date</td>
</tr>
<tr>
<td>Usage limit</td>
<td>100</td>
</tr>
<tr>
<td>Vendor data</td>
<td>*NONE Character value</td>
</tr>
<tr>
<td>Output</td>
<td>*PRINT</td>
</tr>
</tbody>
</table>

---

Some of the parameters on the GENLICKEY command are:
• VNDPWD - this is the software vendor’s password that is encrypted and stored with the product license information. It is the same password that was used on the ADDPRDLIC command when the product was packaged. This ensures that software license keys for your product are only generated by the person (or persons) who know the password.

• SERIAL - this is the serial number of the system for which the license key is being requested.

• PRCGRP - the processor group of the system the license key is being generated for. Use the Work with License Information (WRKLICINF) command on the system the key is being generated for to display the processor group. If you enter a specific processor group, the license key is valid only on that processor group and any lower processor group (or groups). If the customer upgrades the system to a different processor group, the license key becomes invalid. By using this parameter, you can “tier price” your application, that is, you can charge different prices for your application based on the hardware on which it runs.

• Expiration date - this is the date on which the license expires. One possible use of this field is to grant a trial license to a customer. When the customer requests a trial license, you can generate a license key for a small number of users and expire that license in one or two months based on your terms.

• USGLMT - the maximum number of users (usage limit) for the product or feature for which the license key is being generated. For concurrent usage, it specifies the maximum number of jobs allowed to access the product or feature at the same time. For registered usage, this is the maximum number of license users that can be registered to use this product or feature. This limit overrides the default usage limit that was specified on the ADDPRDLIC command for the duration of the license key.

• VNDDTA - this is an 8-character field that can be used by the software vendor to pass data from the software license key to their product. This field can hold any additional licensing information the vendor selects or remain unused (*NONE indicates no vendor data). This data can be returned to the application by the Retrieve License Information (QLZARTV) API.

• OUTPUT - specifies whether the license key that is generated is displayed in a completion message, printed in a spooled file, or saved in a file. If *LICKEYFILE is selected, the license key information is saved to the file specified on the LICKEYFILE parameter. This file can then be used as a key file for Distributed Systems License Option (DSLO) tapes.

The spooled output from the GENLICKEY command is shown in the following example:

```
File . . . . . . : QSYSPRT
Control . . . . . .
Find . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
Create License Key Page 1
5716SM1 V3R6MO 950929 02/16/96 07:46:56
Input parameters
Product ID . . . . . . . . . : IMLG002
License term . . . . . . . . : V2
Feature . . . . . . . . . . : 5001
Serial number . . . . . . . : 1234567
Processor group . . . . . . : *ANY
Expiration date . . . . . . : *NONE
```
13.5.2.2 The ADDLICKEY Command

This command is run by the customer after a license key has been provided by the software provider, that is, the software provider has run the GENLICKEY command.

The parameters on the ADDLICKEY command correlate to the same parameters on the GENLICKEY command. Once you have received a license key from the software vendor, you should enter all values as they are given to you. The license key parameter is where you enter the 18 characters that comprise the key given to you by the software vendor.

13.5.3 Using the Work With License Information Display

The default usage limit is part of your product. For the compliance types of *OPRATION and *WARNING, the usage limit can be changed once the product is installed using the Work with License Information (WRKLICINF) command. For a compliance type of *KEYED, the usage limit can only be changed by adding a new software license key obtained from the software provider. After you obtain the new license key, you can run the ADDLICKEY command to change the usage limit.
The Work with License Information display allows you to work with the products for which you have licenses:

**WRKLICINF PRDID(1MLG002)**

```plaintext
System serial number . . . . . . . . . . : 1016CAD
Processor group . . . . . . . . . . . : P30

Type options, press Enter.
1=Add license key  2=Change  5=Display detail  6=Print detail
8=Work with license users ... 

<table>
<thead>
<tr>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opt</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Parameters or command

F3=Exit    F5=Refresh    F11=Display Usage Information    F12=Cancel 
F17=Position to    F23=More options
```

Some of the things you can do from this display are:

- Use option 1 to run ADDLICKEY for products using *KEYED compliance.
- Use option 2 to run CHGLICINF.
  - For products using *OPRACTION or *WARNING, you can change the usage limit. You can also change a **threshold** value. The threshold value is some number that is less than the usage limit. When this threshold is exceeded, a message can be sent to one or more users. This is a way to find out when you are getting close to exceeding the default number of users, and it may be time to contact your software provider to increase the limit.
  - For products using *KEYED compliance, you can change the threshold value discussed previously.
- Use option 5 to display information about your license usage, including the current number of users of the product and peak usage information as well as the attributes of the product from ADDPRDLICI or ADDLICKEY.
- Use option 8 to see who is currently using the product.

### 13.5.4 Considerations for Use of License Manager

The following decisions must be made prior to enabling a product to use the AS/400 license manager. Product managers as well as marketing, pricing, business planning, and programming should participate in making the decisions.

- Define a use of your product:
  
  You must determine what constitutes an access to the product. At each entry point where your product is used, you need to request a use of the license by calling the QLZAREQ API.
In addition, you have to determine the exit points of your product so you can release the use of the license by calling the QLZARLS API.

- Updates to order process:
  Implementation of user-based pricing affects your order process. Some items to consider are:
  - Are customers allowed to switch from processor to user-based pricing and the reverse?
  - How are changes in the usage count handled (going from \( n \) to \( n+x \) users)?
    - Do you use a 1-800 number?
    - Do you use a fax to distribute keys?
    - Do you use the postal system?
    - How is billing initiated?
    - How do customer records get updated?
  - How does the customer know how many users they are entitled to?
    Should product installation and ordering instructions be updated to reflect user-based pricing?
  - How is the software license key delivered: phone, fax, mail, or electronically using SM/400?
  - How long does take to deliver a software license key?

- Define customized messages:
  The product may use the messages shipped with OS/400 or decide to issue its own messages based on the messages returned from each API.

- Processor groups:
  A software license key can be tied to a processor group so that when a machine is upgraded to a new model with a processor group higher than the current processor group, the key is invalid. If the key is tied to a processor group and a model upgrade occurs that changes the process group, the software license key will expire 70 days from when the model upgrade occurred. The processor group field is a parameter on the generate license key command and API as well as on the ADDLICKEY command. You can view the processor group from the WRKLICINF display.

  If you want your product to run on any processor, you can set the processor group equal to *ANY when generating license keys.

13.5.5 Modifying Your Programs to Use License Management

The role of the license manager is to keep track of how many licenses are in use. Compliance enforcement is done by the requesting program, not by license management. You have to modify your application programs to tell license management that the usage is increased or decreased. The usage is increased or decreased by calling the request (QLZAREQ) or release (QLZARLS) license APIs. Based on this information, license management tells the requesting program when the usage limited is reached.

The following CL programs that use the license management APIs can be interfaced with the Mailing List Main Menu CL program to provide this compliance enforcement support:
A license is requested whenever the Main Menu for the Mailing List application is invoked. The CL program LICREQC (that calls the QLZAREQ API) is called by the program that displays the Menu (MLGMNUC). Refer to Figure 104 on page 190 and Figure 106 on page 192.

A license is released when the user exits the Main Menu for the Mailing List application. The CL program LICRLSC (that calls the QLZARLS API) is called when the user exits the menu. Refer to Figure 105 on page 191 and Figure 106 on page 192.

Figure 105 on page 191 illustrates how the LICRLSC CL program uses the Release License (QLZARLS) API to perform this task. Figure 106 on page 192 shows the modification required in the Mailing List application main menu (MLGMNUC) CL program to decrease the usage count.

Tip

To add additional security into your product, you can use handles. There are two types of handles:

- The Product License Information Handle - this handle is a 16-character string that is given in a message (in your joblog) when you run the ADDPRDLICI command to set up *KEYED compliance. The handle should be encrypted and stored as a constant in your product. At execution time, a product can verify that the product’s license management attributes are the same as when originally packaged prior to requesting a use of the license. To verify the handle, call the QLZARTV API, which creates the handle from some of the attributes in the product definition. Your program can then compare the generated handle against the handle stored as a constant in your product. If they do not match, your application can return without requesting the use of a license.

- The License User Handle - this handle is an 8-character string generated by a product and passed to the license manager when a license is requested using the QLZAREQ API. It is stored in the internal license management object. The same 8-character field must be used by the release license API, QLZARLS, in order for the license to be released. It is suggested that this handle be created by encrypting some piece or combination of data, such as user Signon, product ID, or device ID that can be re-created when the time comes to release the license.
Figure 104. LICREQC CL Program to Request a License Usage Through the Request License (QLZAREQ) API
SOURCE FILEMLG.DEV2/QCLSRC
MEMBER LICRLSC
SEQNBR*...1...2...3...4...5...6...7...8...9...0
100 /*******************************************************************/ 11/16/93
200 /* Standalone licence release program. */ 11/16/93
300 /* Call this pgm from whichever point you want the concurrent */ 11/16/93
400 /* user count decremented. */ 11/16/93
500 /* No parameter is passed to the Release License (QLZARTV) API. */ 11/16/93
600 /*******************************************************************/ 11/16/93
700 PGM 11/16/93
800 /* Declare release parameters */ 11/16/93
900 DCL VAR(&RLPARM1) TYPE(*CHAR) LEN(17) /* Product + */ 11/16/93
1000 DCL VAR(&RLPARM2) TYPE(*CHAR) LEN(8) /* Product + */ 11/16/93
1100 DCL VAR(&RLPARM3) TYPE(*CHAR) LEN(10) /* License + */ 11/16/93
1200 DCL VAR(&RLPARM4) TYPE(*CHAR) LEN(8) /* License + */ 11/16/93
1300 DCL VAR(&RLPARM5) TYPE(*CHAR) LEN(136) /* Error + */ 11/16/93
1400 DCL VAR(&BLANK136) TYPE(*CHAR) LEN(136) 11/16/93
1500 DCL VAR(&RTNVAL) TYPE(*CHAR) LEN(1) 11/16/93
1600 DCL VAR(&MSGDTA) TYPE(*CHAR) LEN(120) 11/16/93
1700 DCL VAR(&MSGID) TYPE(*CHAR) LEN(7) 11/16/93
1800 /* Monitor for all messages (global ignore) */ 11/16/93
1900 MONMSG MSGID(CPF0000) 11/16/93
2000 /* Set error code field to blanks */ 11/16/93
2100 CHGVAR VAR(&RLPARM5) VALUE(&BLANK136) 11/16/93
2200 /* Assign product identification */ 11/16/93
2300 CHGVAR VAR(&RLPARM1) VALUE(MLG002V2R1M05) 11/16/93
2400 /* Assign product identification format name */ 11/16/93
2500 CHGVAR VAR(&RLPARM2) VALUE(LICP01) 11/16/93
2600 /* Assign license information */ 11/16/93
2700 CHGVAR VAR(&RLPARM3) VALUE(*JOB) 11/16/93
2800 /* Assign license identification format name */ 11/16/93
2900 CHGVAR VAR(&RLPARM4) VALUE(LICL01) 11/16/93
3000 /* Assign number of bytes available for error code */ 11/16/93
3100 CHGVAR VAR(%SST(&RLPARM5 1 4)) VALUE(X'00000078') 11/16/93
3200 /* Call the release license API */ 11/16/93
3300 CALL PGM(QSYS/QLZARLS) PARM(&RLPARM1 &RLPARM2 + 11/16/93
3400 &RLPARM3 &RLPARM4 &RLPARM5) 11/16/93
3500 ENDPGM 11/16/93
3600 ** ** ** END OF SOURCE ** ** **

Figure 105. LICRLSC CL Program to Release a License Usage Through the Release License (QLZARLS) API
Figure 106. Modification to the Mailing List Application Main Menu (MLGMNUC) CL Program to Include License Request and License Release Programs.
13.5.6 Distribution of License Keys Scenario

In this section, we provide a scenario of how license keys are generated and distributed.

A software provider, ABC, has created a product that is licensed using keyed compliance. Company XZY who has 50 AS/400 systems, some at remote sites, decides to purchase the product from ABC and install the product on all 50 systems. The following steps are done:

At company ABC:
1. Package the product.
2. Run the ADDPRDLICl command to set up keyed compliance for the product.
3. Obtain the serial numbers of all 50 systems from company XYZ.
4. Run the GENLICKEY command 50 times, once for each of XYZ’s 50 systems. On the GENLICKEY command, specify *LICKEYFILE for the output parameter. Specify the same output file and member and *ADD in the LICKEYMBR parameter. The result of this is that all of the license keys for company XYZ are stored in this output file.
5. Run SAVLICPGM to tape.
6. Use CPYTOTAP to copy the license key file to the same tape.
7. Ship the tape to the central site location of company XYZ.

At company XYZ:
1. Run RSTLICPGM to restore the product on the service provider system.
2. Run the ADDLICKEY command with the LICKEYINP parameter set to *TAPE. This step copies the license keys for all 50 systems into the license key repository on the service provider.
3. Run the SNDPRD command to distribute the product to the 49 service requester systems. Set the SNDLICKEY parameter to *YES. The license key is sent along with the product and is installed automatically on each service requester.

For the preceding scenario, if the license keys need to be updated for the product on the service requesters at a later time, the process at the central site system is:
1. Obtain the new license keys from the software vendor and use the ADDLICKEY command to store the keys in the license key repository on the service provider.
2. Use the ADDLICCRQA command to distribute the keys to the service requesters. This process assumes that MSS/400 is installed on the service requesters.

13.6 Using SystemView System Manager/400 to Manage Software Changes

In Chapter 8, “Servicing Your Products With the AS/400 PTF Process” on page 87, we discussed how to use Application Development Manager and SM/400 to service your product. In this section, we discuss the capabilities of SM/400 to act as a central site processor of software problems and a distribution center of PTFs in this topic. This leads to the SM/400 service provider support.
With SM/400 service provider support as a base, subsequent topics discuss AS/400 problem log processing and the capability for user applications to include IBM-like problem information (First Failure Data Capture (FFDC)). Using these facilities can speed up and automate problem identification and PTF distribution.

To satisfy their customer service requirements, the XYZ Company decided to implement the service requester and service provider concepts of SM/400. By packaging their application into a product, they enabled it for maintenance using the AS/400 PTF process. Now the company is ready to go one step further and automate their service process using other SM/400 maintenance attributes. We do not cover all of the features in SM/400, but use only some of them to illustrate our scenario. For a complete description of the product features and functions, refer to the System Manager/400 Use.

XYZ Company decided to configure the central site system as the service provider. This system provides service support (problem and PTF management) for other systems in the network.

The remote systems are configured as service requesters and they can request service from the service provider. Service requester support is part of OS/400. SM/400 must be installed only at the central service provider site.

Some of the advantages of using one system at the central site to perform service provider functions are:

- No separate communication equipment is needed on each remote site to connect to IBM through ECS.
  
  This reduces cost and assists in tracking software levels on each AS/400 system in the network.

- The service provider can act as a PTF repository and distribution center.
  
  This can assist in tracking the software PTF levels on each AS/400 system in the network.

Figure 107 shows a typical network configuration where SystemView System Manager/400 service provider support is used to support several AS/400 service requesters. The AS/400 service provider can provide several services directly to the service requesters and act as a single interface to IBM-provided service support.
The service provider location is typically associated with the Help Desk or the Head Office location where competence in network and AS/400 system operations is high.

Service Providers define the services to which their requesters are entitled. They receive, process, and respond to service requests, and can distribute PTFs for IBM products and non-IBM products that have been packaged and defined to the system.

A Service Requester can request help for hardware, software, or other problems. It can collect information in a problem record for escalation and further analysis. It can send requests for and receive PTFs.

The provider (or providers) are the only systems that require SystemView System Manager/400 since requester functions are already contained in OS/400.

In a multiple application support environment, it is possible to have multiple service providers and to spread relevant skills and product knowledge over more than one machine. In this situation, with MSS/400 installed, Service Requesters can direct specific requests to the relevant service provider, thus presenting a single interface to the end users for problem determination and escalation. This allows for better control of personnel, software, and hardware. In more complex networks, some AS/400 systems may act as both a provider and a requester.

13.6.1 PTF Support on the Service Provider

The following functions are available on the service provider to enable it to act as a PTF repository and distribution center:

- Order groups of IBM Licensed Program Products PTFs by specifying the product number on the Order Supported Products command.

  First, you must add the product to the supported products list at the service requester using the Work with Supported Products command (WRKSPTPRD). Second, you must add the product to the Order Supported Product PTFs
(ORDSPTPTF) command. This command provides a “give me what I need” capability. When this command is processed, the PTFs that are on the service provider, either installed or in save files, are compared with what is available from IBM service support. This command is only supported between the service provider and IBM service support.

- **Receive special PTF notification:**
  When you order PTFs for one or more licensed programs using the ORDSPTPTF command, you may receive a message notification indicating that you have a defective PTF in your system or a high-impact pervasive (HIPER) PTF that is missing from or not applied on your system.

- **Check for prerequisite and corequisite PTFs:**
  When PTFs are received from IBM Service Support, prerequisite and corequisite checking is performed. A prerequisite PTF is one that must be loaded and applied before the PTF that was specifically ordered. A corequisite PTF is one that needs to be loaded and applied at the same time as the PTF that was specifically ordered. Corequisite PTF relationship is only available from IBM Service Support.

- **Create a set of PTFs from PTFs on tapes:**
  You can copy a PTF from tape or CD-ROM (in V3R6, V3R7) to a save file to populate the PTF repository at the service requester for electronic distribution. The CPYPTFSAVF command allows the service provider to copy all of the PTFs associated with a supported or installed product from tape to PTF save files stored in library QGPL.

- **Support systems with different customer-defined products:**
  The service provider system must have the SM/400 *BASE part of a user defined product, but does not need to have all parts or SM/400 options of the product. This enables the service provider to support PTFs for different releases of the product or different languages of the product without having all of them stored on the service provider system.

  Note that the service provider system must be at the highest or later (newer) OS/400 release level of the service requester systems.

- **Create PTF packages:**
  Using the create PTF package command (CRTPTFPKG), the service requester can easily create a PTF package (selected group of PTFs). Rather than specifying individual PTFs, this command operates against all PTF save files for all supported products and accepts an output file PTF list for select and omit control.

- **Control PTF distribution:**
  The service provider Attributes (CHGSRVPVDA command) can control whether PTFs are automatically sent upon a request received from a service requester and:

  Hold PTFs (HLDPTF command).
  Release PTFs (RLSPTF command).
  Delete PTFs (DLTPTF command).

- **Support service provider - service requester over SNA-backbone network:**
  If your AS/400 systems are connected through a Host-centered backbone network, and you are using NetView DM for distribution, you can still use the
software and problem management capabilities of SystemView System Manager/400. In this case, you still use the SystemView System Manager/400 service provider/service requester functions to report problems and find solutions through symptom strings in the problem log entries. However, you hold your PTFs (HLDPTF) to prevent them from being distributed through SNADS and distribute them through NetView DM plans instead.

SystemView Managed System Services/400 supports the “retrieve PTF” and “send PTF” functions with NetView Distribution Manager Version 1 Release 5 on MVS. Earlier releases of NetView DM can communicate with OS/400 DSNX support to achieve these functions but user-written programming is necessary as DSNX does not formally support the “PTF functions”. Further explanation in this area is beyond the scope of this document.

13.6.2 Problem Log Support on the Service Provider

The following functions are available on the service provider through the system problem log support.

• Logging of a remote problem:

A problem record is created in the problem log of the service provider if a service requester:

- Reports a problem to a service provider by way of a service request and the service provider log remote problems (LOGRMTPRB) attribute is set to *SRVRQS or *ALL.

- Sends an alert to the service provider and the log remote problems (LOGRMTPRB) service provider attribute is set to *ALERT or *ALL. The alert must also have an associated problem record on the service requester.

No automatic action takes place if the service requester that sent the problem notification is not found in the list of systems for which you provide service (as defined by the WRKSRVRQS command).

If an operator at a service requester has run local problem analysis and sent the service request, the results of the analysis are included in the service request.

• Automatic scan for existing PTFs:

Any problem data from the service requester is automatically used to search for a match with existing PTFs. If a match is found, the associated PTFs can be automatically sent to the service requester unless “send PTFs automatically” for the requester is set to *NO.

If a PTF match is not found, the problem is written as a problem log entry along with its symptom string information (if any).

• Remote problem analysis:

The service provider can run remote problem analysis in either of two ways:

- Problem record is on both the requester and provider problem logs:

  From the problem record on the service provider system, the operator can run remote problem analysis. The analysis results are stored in both the service provider and service requester problem systems.

- Use the ANZPRB problem command for user-detected problems by specifying ANZTYPE(*REMOTE):
The analysis results are stored in both the service provider and service requester problem records.

13.6.3 Configuring Service Provider - Service Requester Functions

SystemView Systems Manager/400 is dependant on SNADS and APPN, which may already be configured. Regardless of whether SNADS and APPN are being configured or already configured, there are key configuration parameters that need specific values for service requester/service provider functions to work successfully. We present them in this document, but you should refer to the SystemView System Manager/400 User’s Guide for further information. The key SNADS-related parameters that need to be modified from the typical default SNADS configuration for each system are: assigning a secondary name that includes values in both the “name” and “group” parameters, a SNADS configuration routing table entry corresponding to that secondary name, and a System Directory entry with a user ID and address that correspond to the secondary system name. There are also key values that must be entered into the “Work with Service Provider” (WRKSRVPVD command) menu on the service requester system and the “Work with Service Requester” (WRKSRVRQS command) menu on the service provider system. A flowchart and summary of the steps follow.
In this example, the service provider is named SYSNAM01 and the service requester is name SYSNAM02. Refer to Chapter 1 - "Setting up for System Manager/400" and Appendix C - "Example Setup and Configuration of an AS/400 Network Using SNADS in the System Manager/400 Use" for additional help.
Note

In Figure 108, step 3, you should note that if the service requester has Managed System Services/400 installed, you do need to perform the actions in that step. The examples in the rest of this section assume that you do not have MSS/400 installed, and show you how to verify the CFGDSTSRV options and the existence of the correct directory entry.

Review the following Configure Distribution Services and System Directory command examples:

- CFGDSTSRV Option 2, Routing table
- CFGDSTSRV Option 3. Secondary System Name table
- WRKDIR option 1. Add a Directory entry

Service Provider System: Local System Name(SYSNAM01), Local Control Point Name(SYSNAM01), Local Network ID(ITSCNET).

- Routing entry (CFGDSTSRV option 2, Routing Table) for service requester system:

  Display Details of Routing Table Entry
  Destination system name/Group . . . . : SYSNAM02 ITSCNET
  Description . . . . : Link to SYSNAM02 - Group field required by SVSM/400

- Secondary system name (CFGDSTSRV option 3, Secondary System Name Table) for the service provider system:

  Configure Secondary System Name Table
  System: SYSNAM01

  To add, type choices, press Enter.
  To remove, blank system name and group, press Enter.

  --Secondary System--
  Name Group Description
  SYSNAM01 ITSCNET Entry required by SVSM/400

- Directory entry - local service provider user:

  When you use the Work with service requesters (WRKSRVRQS) command to configure the service requesters at the service provider system, the following directory entry is automatically configured. Note the user profile QSRVBAS, which is also automatically created.

  Display Directory Entry Details
  User ID/Address . . . . : SYSNAM01 ITSCNET
  Description . . . . : IBM User for PTF distribution
  System name/Group . . . : SYSNAM01
  User profile . . . . : QSRVBAS
  Network user ID . . . . : SYSNAM01 ITSCNET

- Directory entry - remote service requester user:

  When you use the Work with Service Requesters (WRKSRVRQS) command to configure the service requesters at the service provider system, the following
directory entry is automatically configured for each defined service requester.

Note: Even when the preceding described directory entries are created automatically, they may be accidentally deleted by authorized users and the PTF distribution fails.

Service Requester System: Local System Name(SYSNAM02), Local Control Point Name(SYSNAM02), Local Network ID(ITSCNET).

- Routing entry (CFGDSTSRV option 2, Routing Table) for service provider:

- Secondary System name (CFGDSTSRV option 3, Secondary System Name Table) for service requester:

- Directory entry - local service requester user:

- Directory entry - remote service provider user:

When you use the Work with service provider (WRKSRVPVD) command to configure the service provider at the service requester system, the following directory entry is automatically configured. Note the user profile QSRVBAS, which is also automatically created.
Note: Even when the preceding described directory entries are created automatically, they may be accidentally deleted by authorized users and the PTF distribution fails.

13.6.4 Service Provider, Service Requester Setup Tips

- Connection through VTAM/NCP:
  
  If your Service requester systems are connected to the service provider system through a subarea network (VTAM/NCP), under each Physical Unit (PU) in VTAM/NCP representing an AS/400 system in the network, there must be a Logical Unit (LU) with the same name as the corresponding AS/400 CP name. Because PU and LU cannot have the same name, you should take this restriction into account when designing name conventions. For example:

  PU name : PBOSTON
  LU name : BOSTON
  CP name : BOSTON

- Automating service provider/requester configuration:
  
  The command Work with Service Provider (WRKSRVPVD) can only be run interactively on OS/400 V2R3 or earlier releases. You can use one of the three following alternatives to configure the service provider support on the service requesters:

  - Instruct the remote site personnel on how to use the WRKSRVPVD command to configure service provider support.
  - Sign on to the remote systems and configure service provider support from the central site.
  - Configure service provider support on one system and distribute the file QAEDSPI in QUSRSYS to all remote systems.

- Job QESTP in QSNADS subsystem:
  
  On the service requester, a job named QESTP that runs in the QSNADS subsystem is automatically started each time the QSNADS subsystem is started after the first service provider (other than *IBMSRV) is defined. You may need to stop and restart this subsystem to enable QESTP.

- Job QECS in QSYSWRK subsystem:
  
  On the service provider, a job named QECS that runs in the QSYSWRK subsystem is automatically started when QSYSWRK subsystem is started. This job is needed to receive service requests from service requesters. You can stop and restart this subsystem to enable QECS. Or, you can start the job using the STRSYSMGR command if MSS/400 is installed.

- Remote problem determination and display pass-through to the remote system:
Remote problem analysis under service provider support implicitly invokes the OS/400 Display Station Pass-Through (DSPT) support. You must verify that the system value QRMTSIGN is set correctly (‘VERIFY).

- Optional SNA Alert processing:
  In a service provider/service requester network, it is common to have Alert support configured such that the service provider system is a focal point with the service requesters in its Sphere of Control.
  You may need to Work with Sphere of Control (WRKSOC) to ensure that alerts are correctly routed.

- SystemView System Manager/400 use of APPN networks and SNA Distribution Services (SNADS) for PTF distribution:
  SM/400 uses SNADS for PTF distribution and it uses an APPN session (Management Services Session) to:
  - Send service requests (SNDRVRQS).
  - Report problems.
  - Answer problems.
  - Send a response to the service requester who sent a service request.
  - Order PTFs.

The following figure shows the dialogue that takes place between the service requester and service provider systems. Only the flow PTF sent between the service requester and the service provider uses SNADS; the other flows use the APPN Management Services Session.

![Figure 109. Flows Between Service Requester and Service Provider]

Based on the SNA flow shown in Figure 109, you should test your service provider/service requester setup using the SNDSRVRQS command. With SNDSRVRQS you are using only the APPN session. This does not verify your SNADS configuration. For example, if you open a problem at the service provider using the Send PTF Order (SNDPTFORD) command from a service requester and the PTF is not sent, check your SNADS configuration.

When MSS/400 is not installed on the service requester, the type of SNADS distribution queues that are used for PTF distribution are *SNADS queues. When MSS/400 is installed, SNADS *SVDS (SystemView distribution services) queues are used by default for PTF distribution.
You can use SM/400 to report problems, track problems, rediscover problems (find PTFs associated with a problem log entry through matching symptom strings), run remote problem analysis, simplify your PTF management using the SM/400 capabilities discussed in 13.6, “Using SystemView System Manager/400 to Manage Software Changes” on page 193, and yet use a distribution tool other than SNADS, for example NetView DM, in a subarea network.

The AS/400 system to AS/400 system service requester/service provider relationship enables the AS/400 service provider (central site at XYZ Company) to offer the same service that it is getting from IBM service support customized to the service requesters needs plus a seamless support for XYZ Company’s own products.

We have just discussed how SM/400 Service Provider and Service Requester functions can be used to manage user applications packaged as products. The following sections provide more details on how problem data is processed at the Service Provider and how adding IBM First Failure Data Capture information to your application can assist in speeding up the problem determination process and identification of existing PTFs.

13.6.5 Connecting a PTF to a Problem Symptom String

A PTF can be associated to one or more problem log entries that, in turn, contain a recorded symptom string. The link between the PTF and the problem log entry/symptom string takes place when the PTF is created and, therefore, the problem log entry must exist before the PTF is created. Once a PTF is linked to a problem log entry/symptom string, the next time a service requester system reports a problem with a matching symptom string, the PTF is identified (problem rediscovery) and can be automatically distributed to the service requester if SNADS/ODF is being used for distribution; otherwise the identified PTF can be retrieved and distributed through, for example, NetView DM.

You can create multiple problem log entries/symptom strings to account for the different ways the same defect might be reported and assigned the multiple problem log entries to the same PTF for problem rediscovery and PTF retrieval.

To associate a PTF to one or more problem symptom strings:

1. At the service provider (where the PTF repository resides and where the PTF is created), create the problem log entry (or entries) containing the problem symptom string.
2. Assign the PTF to the Problem Log Entry previously created.
3. Create the PTF that was assigned to the problem log entry/symptom string.
4. The next time a service requester reports a problem with a symptom string associated to the PTF, the PTF is identified and automatically sent to the requester.

Scenario Description: In the next sections, we focus on problem reporting and linking the PTF to the problem log entry/symptom string. To keep focus on these two items, we move apart from the overall PTF development cycle and just concentrate on the following scenario:

The problem being used is that the command USCRTUSCMD fails with the following messages:
To keep our focus, we do not need (or want) to know the purpose of the USCRTUSCMD, what the problem is, and how to fix it. At this point, we are only interested in collecting the error information (error messages, program name, and so on) to create and report the problem.

13.6.5.1 Using ANZPRB to Create Problem Log Entry/Symptom String

Let’s assume for the purpose of our discussion that a developer discovers the problem after the product has already been distributed and installed in the field. Another alternative is that a user discovers the problem before the developers do and reports the problem.

The fix to the problem is being developed and the corrected objects are exported to the PTF development library. Before the PTF is created, we need to create the problem log entry and associated symptom string for future problem rediscovery.

Using the Analyze Problem (ANZPRB) command, a series of displays is presented for entering information to identify the problem. The System Manager/400 Use contains detail information on this subject. We summarize here the steps to create a problem log record with an associated symptom string using ANZPRB:

1. Type “ANZPRB” and press the Enter key.
2. Type “1.” This AS/400 or attached devices, and press the Enter key.
3. Type “1.” Job or program problem (application or system), and press the Enter key.
4. Type “1.” Creates incorrect output, and press the Enter key.
5. Type “2.” Incorrect output, and press the Enter key.
6. Type “2.” Create problem record, and press the Enter key.
7. Type “1.” to select the product, and press the Enter key.
8. Enter the error message information and press the Enter key.

At this point, you are entering the information used to build the symptom string.

9. Type a brief problem description and press the Enter key.

10. Press F3 to return to the Command entry display.

Note: In this example, we are assuming that the problem log entry is being created at the service provider with the sole purpose of connecting a PTF to a symptom string; that is why we stop after the problem entry is created. If you are discovering and reporting the problem, continue at this point with the steps described in Section 13.6.5.4, "Reporting the Problem From a Service Requester" on page 209.
13.6.5.2 Assigning a Symptom String to the PTF

The Create PTF (CRTPTF) command automatically searches the problem log for any symptom strings associated with the PTF ID being created. The symptom strings found are included in the PTF.

When a problem is reported with the same symptom string of an existing problem, the PTF that includes the symptom string is identified, and if you want, automatically distributed to the service requester. If you want to use this problem rediscovery capability of SM/400, you must assign the PTF to problems before you use the Create PTF (CRTPTF) command.

Use the following sequence to assign the next PTF (1MP0007) to the problem log entry and symptom string created using ANZPRB:

1. Determine the next PTF ID to be assigned. The Work with PTF menu includes an option that helps you keep track of PTF IDs.
2. WRKPTF LICPGM(1MLG002) RLS(V1R1M0)
3. Press F19, Change PTF range:
   The next PTF ID available is shown.

4. Use the WRKPRB command to display the list of problems.
5. Select option 8 to Work with the Problem:

6. Select 5 to Answer the problem:
Work with Problem
System: SYSNAM01

Problem ID . . . . . . . : 9406550741
Origin. . . . . . . . . . : ITSCNET.SYSNAM01
Current status . . . . . : READY
Problem . . . . . . . . : Object xxxxxxxxxx type *USRSPC already exists in Library xxxxxxxxxx.

Select one of the following:

2. Report problem
4. Verify problem corrected
5. Answer problem
20. Close problem
30. Save APAR data to APAR library

Selection

5

7. Select 2 to Work with PTF information:

Answer Problem
System: SYSNAM01

Problem ID . . . . . . . : 9406550741
Origin. . . . . . . . . . : ITSCNET.SYSNAM01
Current status . . . . . : READY
Problem . . . . . . . . : Object xxxxxxxxxx type *USRSPC already exists in Library xxxxxxxxxx.

Select one of the following:

1. Answer from PTF database
2. Work with PTF information

Selection

2

8. Select option 1 to ADD the PTF to the problem:

Work with PTF Information
System: SYSNAM01

Problem ID . . . . . . . : 9406550741
Origin. . . . . . . . . . : ITSCNET.SYSNAM01
Current status . . . . . : READY
Problem . . . . . . . . : Object xxxxxxxxxx type *USRSPC already exists in Library xxxxxxxxxx.

Type options, press Enter.
1=Add 4=Remove 8=Work with PTF

Opt PTF ID Requested PTF Save File
1 IMP0007

(No PTFs associated with problem.)

Note: PTF ID *GEN will return the next available PTF ID.

9. Enter the requested information:
Add PTF Entry

Add PTF Entry

Problem ID . . . . . . : 9406550741
Origin . . . . . . . . . : ITSCNET.SYSNAM01
Current status . . . . . : READY
Problem . . . . . . . . : Object xxxxxxxxxx type *USRSPC already exists in Library xxxxxxxxxx.

Type choices, press Enter.

PTF ID . . . . . . . . . 1MP0007 Character value, *GEN
Product ID . . . . . . . 1MLG002 F4 for list
Release . . . . . . . . . V1R1M0 VxRxMx

13.6.5.3 Create the PTF

We are assuming that the problem was found in the USCRTUSRPG program and the fix was developed using Application Development Manager and exported to the MLGPTFLIB. Now we are ready to create the PTF:

```
PMM
CRTPTF PTF(1MP0007) LICPGM(1MLG002) RLS(V1R1M0) +
   OPTION(*BASE) LODID(*CODEDFT) +
   PTFOBJ((USCRTUSRPG *PGM)) +
   OBJLIB(MLGPTFLIB *PRINCIPAL) +
   COVER((MLGPTFLIB/QTXTSRC CLV1MP0007 2924))
ENDPGM
```

Once loaded and applied, the PTF is installed in the product’s principal library for option *BASE, load *CODEDFT.

13.6.5.4 Reporting the Problem From a Service Requester

A user at a service requester runs into the problem and reports the problem to the service provider using the Analyze Problem command (ANZPRB) as described in Section 13.6.5.1, “Using ANZPRB to Create Problem Log Entry/Symptom String” on page 205, but the user at the service requester reports the problem to the service provider after opening the problem record at the local system. This is the sequence of steps that takes place after the user at the service requester creates the problem log entry:

1. The user chooses not to save problem data:

```
Save Problem Data

A PTF for the problem was not found on the local system. If you have APAR data to be saved, you may want to save this data now. This data may be requested if you report the problem and a PTF cannot be found.

Type choice, press Enter.

Save APAR data . . . . . . N Y=Yes, N=No
```

2. Prepare a service request to report the problem:
The problem has been logged.

Select one of the following:
1. Prepare service request
2. Send alert
3. Send alert and prepare service request

Selection 1

3. Enter contact information in case the service provider needs to call back:

<table>
<thead>
<tr>
<th>Company</th>
<th>Brice Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact</td>
<td>Kalianne Morrison</td>
</tr>
<tr>
<td>Mailing address</td>
<td>Building 663-3</td>
</tr>
<tr>
<td>Street address</td>
<td>3605 Highway 52N</td>
</tr>
<tr>
<td>Building</td>
<td>Rochester</td>
</tr>
<tr>
<td>City/State</td>
<td>MN</td>
</tr>
<tr>
<td>Country</td>
<td>USA</td>
</tr>
<tr>
<td>Zip code</td>
<td>55901</td>
</tr>
<tr>
<td>Telephone numbers</td>
<td></td>
</tr>
<tr>
<td>Primary</td>
<td>507-286-0000</td>
</tr>
<tr>
<td>Alternative</td>
<td></td>
</tr>
<tr>
<td>National language version</td>
<td>2924</td>
</tr>
</tbody>
</table>

4. Select problem severity:

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>9406558693</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current status</td>
<td>READY</td>
</tr>
<tr>
<td>Problem</td>
<td>User space already exists</td>
</tr>
</tbody>
</table>

Select one of the following:
1. High - Requires immediate solution
2. Medium - Restricts function
3. Low - Limits function
4. None - Operates with full function

5. Select the service provider to report this problem to:
6. Send the request to the service provider:

```
> anzprb
No PTFs found that match symptom for problem 9406558693.
PTF for problem not found on local system.
Sending service request 9406558693.
PTF 1MLG002-1MP0007 V1R1M0 ordered.
1 PTF(s) ordered for problem 9406558693.
```

**Note:** The PTF is sent automatically from the service provider to the service requester if the PTF save file is released at the service provider and the service provider attribute `Send PTFs automatically` is set to `*YES`. 

7. The PTF to fix the problem with the symptom string entered at the service requester is found at the service provider and the following shows the messages returned to the service requester:

```
> anzprb
No PTFs found that match symptom for problem 9406558693.
PTF for problem not found on local system.
Sending service request 9406558693.
PTF 1MLG002-1MP0007 V1R1M0 ordered.
1 PTF(s) ordered for problem 9406558693.
```

**Note:** The PTF is sent automatically from the service provider to the service requester if the PTF save file is released at the service provider and the service provider attribute `Send PTFs automatically` is set to `*YES`. 

### Change Service Provider Attr (CHGSRVPVDA)

<table>
<thead>
<tr>
<th>Type choices, press Enter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send PTFs automatically ....</td>
</tr>
<tr>
<td>Message queue .............</td>
</tr>
<tr>
<td>Library ....................</td>
</tr>
<tr>
<td>Maximum distribution ......</td>
</tr>
<tr>
<td>Hold PTFs received .........</td>
</tr>
</tbody>
</table>
13.6.6 Developing Your Own PTF Inventory

SystemView System Manager/400 does not keep an inventory of all PTF activity across a network of AS/400 systems. This inventory, however, may be obtained for each AS/400 system and maintained on a central AS/400 system. The redbook, Managing AS/400 Networks with Operations Control Center/400, contains an example of how this can be achieved.

13.7 First Failure Data Capture (FFDC)

First Failure Data Capture (FFDC) is a function that a program can invoke when it detects a “PTFable” problem:

- FFDC logs the problem. Once logged, the problem can be reported to a service provider.

- FFDC builds a symptom string:
  The symptom string is sent to the service provider when the problem is reported. The service provider searches symptom strings associated with existing PTFs for a match. If it finds one, it sends the PTF to the AS/400 system that reported the problem.

- FFDC collects problem determination data:
  If a PTF does not exist, the developer is notified. The developer uses the data to resolve the problem.

  It is important that the data that is collected is sufficient to resolve the problem. The developer should not have to ask the user for more data, or worse, ask the user to recreate the problem.

A serviceable program decreases costs and increases customer satisfaction. IBM uses FFDC in its own software.

13.7.1 Definitions and Terminology

Detection ID: This ID is up to 12 characters that uniquely identifies the problem. It can be a message ID, a return code, a parameter value, and so on. FFDC puts the detection ID in the symptom string.

Point of Failure: This is a number that identifies an instruction, a block of code, or a subroutine where the problem was detected. It must not be an instruction number or offset. FFDC puts the point of failure value in the symptom string.

The point of failure value does not always pinpoint the exact location at which a problem was detected, but it can be used to narrow the range of possible locations.

QSFWERRLOG: This is a system value used to enable or disable FFDC. By default, FFDC is enabled (value *LOG).

Probe: This is the point at which a program detects a problem.

Reporting program: This is the program that reports the problem. It might not be the same as the detecting program or suspected program.
**Suspected program:** This is the program that has the problem. It might not be the same as the detecting program or reporting program. FFDC puts the name of the suspected program in the symptom string.

**Detecting program:** This is the program that detects the problem. It might not be the same as the suspected program or reporting program.

**Symptom string:** This is the character string that identifies a problem and its PTF. FFDC builds it for you.

The symptom string consists of:

- The name of the program suspected of having the problem.
- Optionally, the name if the ILE module suspected of having the problem.
- Optionally, the point of failure.
- Optionally, a detection ID.

The format of the symptom string (for V3R6M0) is:

- 5716 F/(suspected module / ILE module) MSGXXXYYYY RC(point of failure)

Symptom string ...: 5716 F/OBGENERR1 MSGCPF9870 RC5

### 13.7.2 Guidelines for Using FFDC

1. Identify PTFable situations that include, for example:
   - Unexpected return codes from another program.
   - 9XXX major/minor return codes from Data Management.
   - Function checks.

   The section of the program where the potential PTFable situation might arise can be associated with a given point of failure value.

2. For each PTFable situation, determine which program or ILE module has the problem:
   - If your program receives unexpected input parameters, your program should inform the caller and it should decide if FFDC should be invoked.
   - If FFDC is being invoked for a function check, an escape message that indicates a PTFable situation, or an unexpected return code, your program has the problem.
   - If FFDC is being invoked in the central error handling routine of the program (such as an external exception handler), the program that was running when the problem occurred has the problem.

3. For each PTFable situation, determine what data to collect:

   The program must collect enough data to resolve the problem so that the developer does not have to ask the user for more data, or worse, ask the user to re-create the problem. Collect data such as:

   - Input parameters
   - Return codes
   - Structures
   - Objects
   - Joblog

4. For each PTFable situation, determine the detection ID:

   - If FFDC is being invoked for an unexpected return code, the detection ID is the return code.
If FFDC is being invoked for a message that is known to be PTFable, the detection ID is the message ID.

5. If you do not want to log problems for exceptions due to damaged objects, pass the message reference key (MRK) of the function check message to FFDC. If the exception was due to a damaged object, FFDC does not log the problem.

6. Assign point of failure to:
   - Subroutines
   - Major blocks of logic
   - Instructions that check for unexpected values

   The point of failure must be unique within a program or ILE module.

**13.7.3 Log Software Error (QPDLOGER) API**

The Log Software Error (QPDLOGER) API allows a program to report a software problem and provide the data needed to resolve the problem. When the API is called, any error data provided is spooled to one or more spooled files, a symptom string is created, an entry is created in the problem log, and a message is sent to QSYSOPR message queue indicating that a software error has been detected. If the system is a service requester, the problem is reported to the AS/400 service provider that is the alert focal point for that service requester. For more information on the QPDLOGER API, refer to the *System Programmer’s Interface Reference* manual.

**13.7.3.1 A Tool to Simulate Error Reporting (OBGENERROR)**

The command OBGENERROR and program OBGENERR1 in the following figure (the source is found in file SG244187.1/QSRCCH14) can be used as a tool to familiarize yourself with the AS/400 First Failure Data Capture API (QPDLOGER). The purpose of this tool is to generate an error by sending an "ESCAPE message. The command OBGENERROR provides the user interface to enter the parameters that you want to pass to the FFDC API (QPDLOGER). In a real application environment, those parameters are collected at the point of failure and the API is called by the error recovery routine.

```plaintext
CMD PROMPT('Generate Software Error')
PARM KWD(MSGID) TYPE(*NAME) LEN(7) MIN(1) +
   PROMPT('Message Id to simulate Error')
PARM KWD(MSGF) TYPE(Q1) MIN(1) +
   PROMPT('Message + File')
Q1: QUAL TYPE(*NAME) LEN(10) DFT(QCPFMSG) +
   SPCVAL((QCPFMSG))
   TYPE(*NAME) LEN(10) DFT(*LIBL) +
   SPCVAL(*LIBL) PROMPT('Message File + Library')
PARM KWD(MSGDTA) TYPE(*CHAR) LEN(50) MIN(1) +
   PROMPT('Message Data') / Message Data to +
   pass to the message that simulates +
   software error */
PARM KWD(POF) TYPE(*DEC) LEN(2) PROMPT('Point of +
   Failure')
PARM KWD(JOBLOG) TYPE(*CHAR) LEN(1) RSTD(*YES) +
   DFT(Y) VALUES(Y N) PROMPT('Create Job Log')
```

*Figure 110. Command OBGENERROR*
/* Parameters Passed by CMD OBGENERERROR */
/* Pgm Name OBGENERR1 */
PARM(MSGID MSGFILE MSGDTA POF JOBLOG)

DCL VAR(MSGID) TYPE(*CHAR) LEN(7) /* Message id +
of the message that creates the error */
DCL VAR(MSGFILE) TYPE(*CHAR) LEN(20) /* Message +
File Name and Library */
DCL VAR(&FILNAM) TYPE(*CHAR) LEN(10) /* Message+
file name */
DCL VAR(&FILLIB) TYPE(*CHAR) LEN(10) /* Message +
file LIBRARY */
DCL VAR(MSGDTA) TYPE(*CHAR) LEN(50) /* Message +
data of the message that creates the error */
DCL VAR(MRK) TYPE(*CHAR) LEN(4)

/* Parameters required by QPDLOGER API */
DCL VAR(PGM) TYPE(*CHAR) LEN(10) /* Suspected +
Program Name */
DCL VAR(DETID) TYPE(*CHAR) LEN(12) /* Detection +
Id */
DCL VAR(POF) TYPE(*DEC) LEN(20) /* Point of +
failure (DECIMAL) */
DCL VAR(POFBIN) TYPE(*CHAR) LEN(4) /* +
Point of failure (binary) */
DCL VAR(JOBLOG) TYPE(*CHAR) LEN(1) /* Print job log */
DCL VAR(DTA) TYPE(*CHAR) LEN(60) /* Data Items */
DCL VAR(DTAOFFLEN) TYPE(*CHAR) LEN(8) /* Data +
Items offset and length */
DCL VAR(OFFSET) TYPE(*DEC) LEN(20) VALUE(0) /* +
Data Offset (decimal) */
DCL VAR(OFFSETBIN) TYPE(*CHAR) LEN(4) /* Data +
Offset (binary) */
DCL VAR(LENGTH) TYPE(*DEC) LEN(20) VALUE(0) /* +
Data Length (decimal) */
DCL VAR(LENGTHBIN) TYPE(*CHAR) LEN(4) /* Data +
Length (binary) */
DCL VAR(NDROBJ) TYPE(*DEC) LEN(9) /* Number of +
data Items (decimal) */
DCL VAR(NDROBJBIN) TYPE(*CHAR) LEN(4) /* Number +
of data Items (binary) */
DCL VAR(OBJNAME) TYPE(*CHAR) LEN(10) /* Object +
Name */
DCL VAR(NROBJ) TYPE(*DEC) LEN(9) /* Number +
of object names (decimal) */
DCL VAR(NROBJBIN) TYPE(*CHAR) LEN(4) /* Number +
of object names (binary) */
DCL VAR(ERROR) TYPE(*CHAR) LEN(4) /* error +
code (see structure SPIR manual page 2-9 */
CHGVAR VAR(&FILNAM) VALUE(%SST(MSGFILE 1 10))
CHGVAR VAR(&FILLIB) VALUE(%SST(MSGFILE 11 10))

/* Send message to simulate the problem */
SNDPGMMSG MSGID(MSGID) MSGF(FILLIB/FILNAM) +
MSGDTA(MSGDTA) TOPGMQ(*SAME) +
MSGTYPE(*ESCAPE)

/* Monitor for all messages you wish to simulate */
MONMSG MSGID(CF0000 MCH0000)
RCVMSG MSGQ(*PGMQ) MSGTYPE(*EXCP) +
Keyvar(MRK) MSGDTA(MSGDTA) MSGID(MSGID)
When you use the OBGENERROR command, you are prompted for the information in the command parameters that is passed to the QPDLOGER API called by the program OBGENERR1. The following sequence of steps takes place when you use OBGENERROR to simulate a problem, collect error information, and report it using the QPDLOGER API:

1. Execute the OBGENERROR command:

   ```
   Generate Software Error (OBGENERROR)
   Type choices, press Enter.
   Message Id to simulate Error . . CPF9870 Name
   Message File . . . . . . . . . . QCPFMSG Name, QCPFMSG
   Message File Library . . . . QSYS Name, *LIBL
   Message Data . . . . . . . . . . ABCDEFG QTEMP USRSPC
   Point of Failure . . . . . . . . 05 Number
   Create Job Log . . . . . . . . . Y Y, N
   ```

2. A problem log entry is automatically created in the problem log:

   ```
   Work with Problems
   System: RC
   Position to . . . . . . . Problem ID
   Type options, press Enter.
   2=Change 4=Delete 5=Display details 6=Print details
   8=Work with problem 9=Work with alerts 12=Enter notes
   Opt Problem ID Status Problem Description
   5 9406866671 READY Software problem data for OBGENERR1 has been logged.
   9406861507 READY LIC problem data has been logged for QICGL.
   9406861415 READY LIC problem data has been logged for QICGL.
   ```

3. Select Option 5 to display details:

   The symptom string has been automatically recorded in the problem log entry.
4. Page down until you see the display containing the symptom string:

Display Problem Details
System: SYSNAM01

Problem ID . . . . . . . : 9605038727
Origin . . . . . . . . . : ITSCNET.SYSNAM05
Current status . . . . . : READY
Problem . . . . . . . . . : Software problem data for OBGENER1 has been logged. Refer to help text for addition.
Problem message ID . . . . . . . : CPI93B9
Problem type . . . . . . . . . . : Machine detected
Problem category . . . . . . . : *REPORT
Date and time detected . . . . . . . : 02/19/96 11:16:48
System reference code . . . . . . : SRC900FDC7

Display Problem Details
System: SYSNAM01

Problem ID . . . . . . . : 9605038727
Origin . . . . . . . . . : ITSCNET.SYSNAM05
Current status . . . . . : READY
Problem . . . . . . . . . : Software problem data for OBGENER1 has been logged. Refer to help text for addition.
Detecting program ID . . . . . . :
Version . . . . . . . . . . . . :
Release . . . . . . . . . . . . :
Modification level . . . . . . :
Program . . . . . . . . . . . . : OBGENER1
Symptom string . . . . . . . . . . : 5716 F/OBGENER1 MSGCPF9B70 RC5

APAR library . . . . . . . . . : QSC5038727

Display Spooled Files
System: SYSNAM01

Problem ID . . . . . . . : 9406866671
Origin . . . . . . . . . : ITSCNET.SYSNAM01
Current status . . . . . : READY
Problem . . . . . . . . . : LIC problem data has been logged for OBGENER1
Refer to help text for addition

Type options, press Enter.
5=Display
Opt  File   File Nbr   Job  User  Number
5  QPSRVDMP   0019   P23XK76D   ABCDEFG   039366
5  QPJOBJLOG   0020   P23XK76D   ABCDEFG   039366
5  QPDSPJOB   0021   P23XK76D   ABCDEFG   039366

Chapter 13. Advanced topics 217
The associated spooled files are listed here, but QPSRVDMP data is not important as part of this example.

### 13.7.3.2 Inserting First Failure Data Capture Probes in Your Programs

After introducing the concepts and benefits of First Failure Data Capture and how to use the AS/400 Log Software Error API, we revisit our scenario in Section 13.6.5, “Connecting a PTF to a Problem Symptom String” on page 204.

At a high level, the flow of the problem and the objects involved are:

```
  USCRTUSCMD 1 (*CMD)
    USCRTUSRPG 2 (RPG *PGM)
      ....
      Error subr 3 USERROR (CL *PGM) Returns error messages
    *CANCL 4
```

- **1** The command USCRTUSCMD calls its Command Processing Program USCRTUSRPG.
- **2** The command Processing Program USCRTUSRPG fails; the program error subroutine takes control and calls USERROR (CL Program).
- **3** USERROR is a standard error message handling CL program; it receives and returns all of the diagnostic messages and then the escape message, and gives control back to USCRTUSRPG.
- **4** USCRTUSRPG ends.

The RPG program USCRTUSRPG was modified to use First Failure Data Capture. The new program USCRTFFDC is a duplicate of USCRTUSRPG with additional statements added to support FFDC as shown:
We have made simple changes to the programs to enable First Failure Data Capture to collect error information and report the problem. We made the following changes:

1. Added a Point of Failure (POF) field to USCRTFFDC in the sections of the program that might experience a problem. Those sections are:
   - The input parameter section; there is a possible problem if the calling command or program does not pass the right number and type of parameters. We assigned the value “1” to the POF field in this section.
   - The call to the USCRTUS API section; the API might fail to create the User Space. We assigned the value “2” to the POF field in this section.

2. The field Point of Failure (POF) is passed to the error handling program USERIFFDC called by the error subroutine.
```plaintext
PGM PARM(&POF) /SF590000
DCL VAR(&MSGID) TYPE(*CHAR) LEN(7)
DCL VAR(&MSGDTA) TYPE(*CHAR) LEN(256)
DCL VAR(&ERR) TYPE(*LGL) VALUE(0) /* Set to +1 if error */
/* Parameters required by QPDLOGER API */
DCL VAR(&PGM) TYPE(*CHAR) LEN(10) /* Suspected Program Name */
DCL VAR(&DETID) TYPE(*CHAR) LEN(12) /* Detection Id; uniquely identifies the problem */
DCL VAR(&POF) TYPE(*DEC) LEN(20) /* Point of failure (DECIMAL) Identifies the section of the calling program experiencing the problem */
DCL VAR(&POFBIN) TYPE(*CHAR) LEN(4) /* Point of failure (binary) */
DCL VAR(&JOBLOG) TYPE(*CHAR) LEN(1) /* Print job log; Generates spooled files */
DCL VAR(&MRK) TYPE(*CHAR) LEN(4) /* Message reference key to avoid reporting problems caused by damaged objects */
DCL VAR(&DTA) TYPE(*CHAR) LEN(60) /* The information here is spooled to help to diagnose the problem - Data Items */
DCL VAR(&DTAOFFLEN) TYPE(*CHAR) LEN(8) /* Data Items offset and length */
DCL VAR(&OFFSET) TYPE(*DEC) LEN(20) VALUE(0) /* Data Offset (decimal) */
DCL VAR(&OFFSETBIN) TYPE(*CHAR) LEN(4) /* Data Offset (binary) */
DCL VAR(&LENGTH) TYPE(*DEC) LEN(20) VALUE(0) /* Data Length (decimal) */
DCL VAR(&LENGTHBIN) TYPE(*CHAR) LEN(4) /* Data Length (binary) */
DCL VAR(&NBROBJ) TYPE(*DEC) LEN(9 0) /* Number of object names (decimal) */
DCL VAR(&NBROBJBIN) TYPE(*CHAR) LEN(4) /* Number of object names (binary) */
DCL VAR(&OBJNAM) TYPE(*CHAR) LEN(10) /* Object Name */
DCL VAR(&ERROR) TYPE(*CHAR) LEN(4) /* error code (see structure SPIR manual page 2-9) */
MONMSG MSGID(CPF0000 MCH0000 RPG0000 CBL0000) /* Monitor for all the messages you expect to get */
ERROR: RCVM SGMSG PGMO(*PRV) MSGTYPE(*DIAG) MSGDTA(&MSGDTA) MSGID(&MSGID)
IF COND(&MSGID *EQ '0') THEN(GOTO CMDLBL(ERROR_ESC))
SNDPGMSG MSGID(&MSGID) MSGF(QCPFMSG) MSGDTA(&MSGDTA) TOPGMO(*EXT) TOMSGQ(*SYSOPR) MSGTYPE(*DIAG)
MONMSG MSGID(CPF0000 MCH0000 RPG0000 CBL0000) /* Monitor for all the messages you expect to get */
GOTO ERROR
```

Figure 113 (Part 1 of 2). Program USERFFDC Modified to Include Point of Failure Field
Figure 113 (Part 2 of 2). Program USERFFDC Modified to Include Point of Failure Field

We made the following changes to the error handling program:

- **1** Accept the parameter Point of Failure (POF) from the calling program.
- **2** Add variables required by QPDLOGER API.
- **3** Call QPDLOGER API to automatically collect problem information and report the problem.

After the modifications previously described, we created the error situation by running the USCRTUSCcmd twice in a row with the same parameters and without deleting the user space between invocations of the command.
Create User Space (USCRTUSCMD)

Type choices, press Enter.

User Space: . . . . . . . . . . > ABC  Name
Library Name: . . . . . . . . . > QTEMP  Name, *CURLIB
Extended Attributes . . . . . . > ABC  Name
Size of User Space: . . . . . > 150
Initialize User Space To: . . . > 0  Character value
AUTHORITY . . . . . . . . . . > *ALL  *CHANGE, *ALL, *USE, *EXCLUDE
Text Description: . . . . . . . > TEST

A problem log entry is automatically created for this problem with the following symptom string:

Display Problem Details

Problem ID . . . . . . . : 9605045037
Origin . . . . . . . . . : ITSCNET.SYMNAM01
Current status . . . . . : READY
Problem . . . . . . . . : Software problem data for USCRTFFDC has been logged.
Detecting program ID . . . . . : USERFFDC
Version . . . . . . . . . . : 5716 F/USCRTFFDC MSGCPF9870 RC2
Release . . . . . . . . . . . : QSC5045037
Modification level . . . . . : More...
Program . . . . . . . . . . . : USERFFDC
Symptom string . . . . . . . . . : 5716 F/USCRTFFDC MSGCPF9870 RC2

APAR library . . . . . . . : QSC5045037

Press Enter to continue.

F3=Exit  F6=Display problem history  F11=Display APAR library  F12=Cancel
F24=More keys

The spooled file QPSRVDMP associated with this problem shows the following information:

5716SS1 V3R6M0 950929  AS/400 DUMP  014984/ABCDEF
DUMP TAKEN FOR DETECTED ERROR
.SUSPECTED-
..LICENSED PROGRAM-
..FUNCTION-
..LOAD-
..PTF-
.DETECTOR-
..LICENSED PROGRAM-
..FUNCTION-
..LOAD-
..PTF-
..SYMPTOM STRING-
13.7.4 First Failure Data Capture Tips and Hints

- There are internal objects on the system that do not allow the creation of a symptom string that matches one of the last 10 for the job and one of the last 50 for the system. This restriction is valid even if you delete the problem log entry.

- The QPDLOGER API is generally available in OS/400 V2R3 and subsequent releases.

- Symptom strings created with the ANZPRB command do not match symptom strings created by QPDLOGER; you must create the problem log entry to be assigned to a PTF using QPDLOGER if you expect the problem to be reported by QPDLOGER.
Chapter 14. Packaging, Distributing, and Servicing Client/Server Applications

A client/server application such as Client Access/400 is a set of programs and objects of which part run or reside on the PC and part on the AS/400 system. The PC code can physically reside on the PC hard drive or on a shared folder on the AS/400 system, in which case the AS/400 system acts as a file server.

Even when the code is developed for two different platforms (PC and AS/400 system), it can be packaged and distributed as a single entity (a product) and maintained under the same PTF process.

The high-level view of the process to manage client/server applications is similar to the one explained in previous chapters for applications that contain only AS/400 libraries and objects:

- Package your application into an AS/400 product.
- Distribute your product.
- Maintain and service your product.

In the first sections of this chapter, we review some basic concepts needed to understand the implementation of our sample scenario later in the chapter.

In this chapter, we discuss how to package and service client code, along with AS/400 code using SM/400 and how to distribute the client code to the desktop using Client Access/400 update function.

14.1 Packaging Structure of Client/Server Applications

From the packaging point of view, the main difference between an application that runs only on the AS/400 system and a client/server application is that the latter includes PC files. On the AS/400 system, PC files reside in documents in folders so your package contains folders with a structure that maps to the PC directory and file structure.

Most PC applications have one root directory, and multiple subdirectories and files under the application’s root directory. This is the recommended directory structure to package the client portion of a client/server application. The PC application root directory maps to a root folder, the subdirectories map to subfolders under the root folder, and the PC files map to documents in the root folder and subfolders. When packaging your application, the folders and subfolders are specified when you create the product load. Folders to be included as part of the product are specified in the folder list parameter (FLRL) on the Create Product Load (CRTPRDLOD) command. The following rules apply for using folders:

- Each product option has only one root folder that maps to the client’s code root directory.
- A folder cannot belong to more than one product option.
- The root folder must be part of the code load (as opposed to language load).
- When specifying subfolders, the complete path from the root to the subfolder must be included.
• Up to 100 folders can be specified for each product load (only one root folder).
• Documents in folders are not specified when a product is created.
• When a product load is packaged, the folders for the load must exist and the user packaging the product load must have *ALL authority to each folder.
• A user must be enrolled in the system distribution directory to perform the packaging function on a product load with folders.

14.2 Distribution of a Client/Server Product

Once packaged, the product is distributed the same way a product containing only AS/400 code is distributed:

• Through tape, saving the product using the SAVLICPGM command and restoring it on the target AS/400 system using the RSTLICPGM command.
• Electronically, saving the product libraries and folders to save files (*SAVF) using the SAVLICPGM command and restoring them on the target AS/400 system using the RSTLICPGM or SNDPRD commands.

The difference between an application with only AS/400 code and a client/server application is that once the client’s code has been distributed to the server (AS/400 shared folder), in most cases part or all of this code is downloaded to the client’s hard drive. There might be some instances, however, that given the characteristics of the application and the performance requirements, there is no need to distribute the client’s code to the desktop and the PC user only needs to receive authority to use the contents of the product in the folder on the server AS/400 system and can then run the programs from the shared folder drive.

In this section, we discuss how to manage the distribution of client code from the AS/400 folder to the PC’s hard drive. In our scenario, we rely on the administration and update functions of Client Access/400 to manage and perform the distribution of software to the clients. We use a similar approach to the one implemented by the product Client Access/400 to distribute release upgrades and PTFs to the clients. We summarize the Client Access/400 concepts needed to understand the rest of this chapter. For a detailed description of the functions and how to implement them, refer to the manuals Client Access/400 Optimized for OS/2 API and Technical Reference, Client Access/400 for Windows 3.1 User Guide, Client Access/400 for OS/2 Setup and Client Access/400 for DOS Ext Memory Setup. Our scenarios were developed using OS/2 2.1 clients.

14.2.1 Client Access/400 Update Function, Trigger, and Package Files

The Client Access/400 update function compares two sets of files and, if they are not at matching levels, updates one set with the contents of the other. Client Access/400 uses the update function to update personal computers when changes are made on the AS/400 server Client Access folder, including updating to a new release of Client Access.

You can also use the update function to update applications other than Client Access/400. Every time you make changes to the client code on the AS/400 shared folder (installing new files, removing files, or changing existing files using, for example, the PTF process), you can use the Client Access update function to ensure that all your PC users are using the same version of the files.
In summary, by using the Client Access/400 update function in combination with the administration function, you can:

- Download new client code on the Client Access administered user’s PC.
- Remove client code from the Client Access administered user’s PC.
- Synchronize files on source and target directories so that the date and time of the files are the same.

For Client Access/400, the installation program creates the appropriate entries in your STARTPCS.CMD and CONFIG.PCS files to apply the updates from the Client Access/400 folder to the user’s Client Access/400 directory. The update function automatically applies the update when you start Client Access/400. The objective is to keep the files in the source directory (shared folder) and target directory (PC’s hard drive directory) at matching levels (date and time). It is important to understand that the update function keeps the target directory in sync with the source directory, regardless of which one has the newer files. Changes made to files on the target directory (PC files) do not affect the source directory but rather get lost since they are overridden by the files in the source directory the next time Client Access update function runs and discovers a mismatch in the date and time of the target and source files.

14.2.1.1 Trigger Files

The source directory is the folder that contains the master copy of the files. All updates are applied from this folder to the PC’s directory, the target directory. The Client Access update program uses trigger files to shorten the amount of time needed to run when there are no files to be updated. When the update function runs, it checks the source and target directories for specific trigger files.

- If the trigger file is found in both directories and the time and date of the files is the same, then the update function assumes that the target directory is current with the source directory and does no other file checking.
- If the time or date of the trigger files is different, or if the trigger file does not exist in the target directory, then all files in the target directory are compared with files in the source directory to determine which (if any) files need to be updated.

When the update function has successfully updated the application files, the trigger file found in the source directory is copied to the target directory.

The trigger file used to apply fixes (PTFs) to PC code is QPTFIDX.

When the update function uses the trigger files and decides that the source directory is not current, the update function checks all the files in the target directory (PC) and compares them to the files in the source directory (shared folder) to determine which ones need to be updated. Therefore, only files already in the target directory are updated; it is not possible to use trigger files to download new files or delete files.

14.2.1.2 Package File

The source directory (folder) may also contain a package file. The package file describes which files you want to exist in your PC. This list of files must follow a certain format, and must be stored on the directory specified as the source for the update command.
Package files are only processed by the Client Access/400 update program when the package file exists in both the source and target directories, and the time or date of the files in the two directories is different.

Package files are identified by having PKG as an extension and the following format:

Package file name = EXAMPLE.PKG

PKGF Book 1
MBRF BOOKMGR.EXE 2
MBRF EPABC492.DLL
MBRF EPABKBKS.HLP
DLTF OLD.HLP 3
REM An example 4

1. A line must contain the identifier PKGF to indicate that this is a package file.
2. MBRF identifies each file (and its path) part of the package that must be downloaded.
3. DLTF identifies each file (and its path) part of the package that must be deleted from the target directory.
4. REM identifies a comment.

14.2.2 Client Access/400 Administration Function

Important Note

The Client Access/400 Administration Function described in this section is only implemented in the OS/2 16 bits and DOS Extended clients. Therefore, this discussion does not apply to other clients.

The Client Access administration function allows a Client Access designated administrator to control the configurations of all Client Access/400 users from the administrator’s own PC. Master copies of each user’s configuration files are stored on the AS/400 system in folders that enable the administrator to change the user’s Client Access configuration files without having to work on the user’s PC.

When the Client Access/400 product is installed on the AS/400 system, a folder named QIWADM is created. When you use the administration function, you create a user configuration for each Client Access/400 user to be administered. These user configurations are actually folders that reside in the QIWADM/USER folder. Any files in this folder are copied to the user’s PC when the Client Access/400 update function runs.
The following figure shows the subfolder QIWSADM/USER/SUE and the configuration files in Sue’s subfolder.

To make the creation of these configurations easier, the administrator can create model configurations that define how a specific set of users want their Client Access options configured on their PC. The administrator creating the configuration files for a user can base the user’s configuration on a model configuration.

In summary, the Client Access/400 administration function allows the administrator to perform the following activities:

- Define model configurations on which to base user configurations.
- Define the configuration files needed to run Client Access/400.
• Add, change, and delete configuration files for each individual user.
• Create a Client Access installation diskette that can be customized for each user’s PC.
• Administer end users on more than one AS/400 system from the same PC.

When the Client Access user becomes an administered user, an ADMN entry is added in the user’s I:QIWSADM\USER\user_name\CONFIG.PCS file. (By default, STARTPCS uses the CONFIG.PCS file to start Client Access, but you can give this configuration file whatever name you want and pass that name to STARTPCS. In our examples, we use CONFIG.PCS, but remember that this can be a different name). The next time the Client Access update function runs, the user’s CONFIG.PCS file in the PC drive is updated. We discuss the ADMN entry later in this chapter. The following is a sample CONFIG.PCS file for a Client Access administered user.

I:\QIWSADM\USER\SUETYPE config.pcs

RTYP CMGR
ADMN I:\QIWSADM\USER\SUETH, C:\PCSOS2
UPDT I:\QIWSOS3, C:\PCSOS2, S,, Client Access/400
LCLN P2618671
MCAO 128,,2
SFLR 1, I,, SYSNAM01
RMTN SYSNAM01

The PCSINFO.ADM file is created at the time the administrator selects the option Location of Client Access functions in the General Options menu when creating or changing a user or model configuration. This file is the file that the update function checks to download the list of files in a package file (file extension PKG) from the AS/400 shared folder to the Client Access/400 administered user’s PC.

The format of the PCSINFO.ADM file is:

DLTP C:\PCSOS2\VPRT.PKG
DLTP C:\PCSOS2\MSG.PKG
ADDP C:\QUICK\QUICK.PKG

1. DLTP identifies the package file that must be deleted from the administered user’s PC. Only files in the package file’s list are deleted.
2. ADDP identifies a package file that does not exist in the administered user’s PC and the .PKG file is added along with all the files listed in the MBRF entries of the package file.

PCSINFO.ADM may contain other entries not associated with package files. These additional entries are not discussed in this book.

14.2.2.1 Transferring the Client Code to the Personal Computer

In this section, we put together the steps that are followed when you use Client Access/400 update and administration functions to download your product client code to the user’s PC.

1. STARTPCS.CMD runs when you start Client Access. By default, the Client Access/400 configuration file that is used to start the Client Access/400 functions is CONFIG.PCS (this configuration file may have any name that must be passed to STARTPCS; if no name is passed, STARTPCS assumes CONFIG.PCS is the configuration file). There is an entry in STARTPCS.CMD to start Client Access update.
2. The user is a Client Access/400 administered user and, therefore, the CONFIG.PCS file contains an ADMN entry:

```
RTYP CMGR
ADMN I:\QIWSADM\USER\SUE,C:\PCSOS2
UPDT I:\QIWSOS2,C:\PCSOS2,S,,,Client Access/400
LCLN P2618671
MCAO 128,,2
SFLR 1,1,,SYSNAM01
RMTN SYSNAM01
```

The update function processes the ADMN entry in the CONFIG.PCS file and looks at the I:\QIWSADM\USER\SUE\PCSINFO.ADM file for the user on the AS/400 server. If that file has changed since the last update, the update function processes PCSINFO.ADM:

- If it encounters a DLTP entry for a package that exists on the PC, it deletes the package and all the files in the MBRF entries (cross-checking is performed to assure that the update function does not delete a file that is on the MBRF of another existing package file).

- Any package file in ADDP entries that does not exist on the PC is added to the user’s PC, along with all of its files in the MBRF entries. This is done in two steps: first a dummy version of the package file is created, then later, when the update function is processing the corresponding UPDT entry in the CONFIG.PCS file, that package file is processed along with the corresponding MBRF’s entries. Keep in mind that for the UPDT entry to process a package file, this file has to exist on both source and target directories. One of the reasons why the ADMN entry is processed before UPDT is so the .PKG file can be added to the target directory.

3. The UPDT entry is processed. A .PKG file exists now on both source and target directories; MBRF entries in I:\QUICKENW\QUICK.PKG are downloaded to C:\QUICKENW.

```
RTYP CMGR
ADMN I:\QIWSADM\USER\SUE,C:\PCSOS2
UPDT I:\QIWSOS2,C:\PCSOS2,S,,,Client Access/400
LCLN P2618671
MCAO 128,,2
SFLR 1,1,,SYSNAM01
RMTN SYSNAM01
UPDT I:\QUICKENW,C:\QUICKENW,,Quicken
```
Notes:
Prior to the process previously described, the following must be done:

1. Create .PKG file in the folder for the client’s code.
2. Add the UPDT entry specifying source and target paths to the user’s CONFIG.PCS file.
3. The entry ADMN must be manually added to the user’s PC CONFIG.PCS file or in the initial installation using customized installation diskettes.

14.2.2.2 Using Entry and Exit Programs
To complete the scenario of installing the client part of your client/server application, let’s take a look at the ENTRY and EXIT programs that you can run when you are installing a new client application or upgrading the release of an existing one. For more information on this subject, refer to Client Access/400 for OS/2 Setup and Client Access/400 for DOS Ext Memory Setup.

When Client Access/400 update function runs, it checks the QPTFIDX trigger files to determine if you are installing a new release. If QPTFIDX exists on the source directory (shared folder) but does not exist on the target directory (PC), or QPTFIDX does exist in both directories but the time and date are different, it looks at the VRM (version release modification) on both. If the VRM values are different, it looks at the PCSUPDT.DAT file on the source directory to determine which ENTRY and EXIT programs to run during the installation (initial installation or new release). So, if you want to run ENTRY and EXIT programs, you must create the PCSUPDT.DAT file on the source directory and specify the name of the entry and exit programs in this file. These programs can perform whatever tasks are necessary to complete the installation, including updating configuration files, creating new directories, or removing old directories. The ENTRY program is run before the update function and the EXIT program is run after the update function. You can specify both (ENTRY and EXIT) programs, only one of them, or not using them at all by not creating the PCSUPDT.DAT file.

You cannot specify a path for the ENTRY and EXIT programs and, therefore, they must exist in the source path.

14.3 Sample Scenario Overview
In the next sections, we show how to package a client/server application and distribute the client code to the desktops using Client Access/400 administration and update functions. Finally, we develop and distribute a fix for this application.

Our special thanks to Prudential Insurance and Financial Services for allowing us to use a Prudential application and their electronic distribution process as a base for the case study discussed in this chapter.

14.3.1 Sample Client/Server Application Package Structure
You design your package structure using the criteria discussed under Packaging Considerations in the System Manager/400 Use book.

The purpose of showing the following package structure is just to illustrate a package with multiple options; some options contain only AS/400 library objects, some contain only AS/400 folders (that, in turn, may contain office documents or PC files), and some options contain a mix of AS/400 library objects and folders.
with client code. Do not try to understand the options from the function point of view; just focus on the mix of libraries and document library objects that make up the package and how this package structure maps to the parameters in the CRTPRDLOD command.

- **Option *BASE**
  - Primary library:  
    - XPRULIB
  - Additional libraries:  
    - XPRUDTA
    - XPRUSRV
    - XDASLIB
  - PC directory structure:

![PC Directory Tree](image)

*Figure 115. Client Code Structure*

Directory path listing

Path: \XAAUPDT

Subdirectories: COMMAND DESCRIBE PRULETRS TOOLBOX

Files: PRU001.PKG QPTFIDX PRUPTF.PKG

Path: \XAAUPDT\COMMAND

Subdirectories: BIN DATA DLL

Files: None
Path: \XAAUPDT\COMMAND\BIN

Subdirectories: None

Files: CMDTGT.EXE
       CMHLP.HLP
       CONSERVE.BAT
       CONSERVE.ICO
       IBMMOUSE.CMD
       RTDEMO2.EXE

Path: \XAAUPDT\COMMAND\DATA

Subdirectories: None

Files: ACCNOKEY.BMP
       AE01.DAT
       AGTASST1.NFO
       CLNDXA.NFO
       CLNDXL.NFO
       NDEMO.MRO

Path: \XAAUPDT\COMMAND\DLL

Subdirectories: None

Files: CMDCOMON.DLL
       CMDLTR.DLL
       CMDMSG.DLL
       CMDRES.DLL

Path: \XAAUPDT\DESCRIBE

Subdirectories: MACROS

Files: None

Path: \XAAUPDT\DESCRIBE\MACROS

Subdirectories: None

Files: BULLET
       CALENDAR
       CHCASE
       CLOSEALL
       CLRSCRN
The preceding listed PC files reside on the client’s hard drive once the package is installed, but they also reside on the AS/400 server in folders mapping the PC directory structure. After the package is installed, the client code is downloaded from the server. On the central site (or packaging
system). You receive the PC files into the AS/400 packaging folders by copying the directories from a PC hard drive to the shared folder or by directly using the AS/400 shared folder to develop the client’s code.

The following displays show a partial view of the client’s directories, subdirectories, and files mapped to the server’s folder, subfolders, and documents.
• Option 1
  Contains documentation (Office Vision/400 documents).
  - AS/400 folder
    . XPRUPHON

• Option 2
  Contains another set of documentation (Office Vision/400 documents).
  - AS/400 folder
    . XPRUHLP

• Option 3
  This option contains the application tutorial (training library).
  - Primary library
    . XPRUTRAIN

14.3.2 Packaging the Sample Client/Server Application

The packaging process is exactly the same one discussed in previous chapters. The following program was used to package the sample application for this chapter.

Note: Pay special attention to the folder and subfolder path specification for the product loads that include client code.

```
PGM
/**************************************************************************/
/*  Program Name : PRU001PGK */
/*  Program Type : CLP */
/*  */
/**************************************************************************/
/* Delete the objects just in case this program was run before */
/**************************************************************************/
DLTMSGF MSGF(XPRULIB/PRUMSGF)
MONMSG MSGID(CPF0000 SMU0000)
DLTPRDFN PRDFN(XPRULIB/PRUPRDFN)
MONMSG MSGID(CPF0000 SMU0000)
DLTPRDLOD PRDLOD(xprulib/prulod)
MONMSG MSGID(CPF0000 SMU0000)
DLTPRDLOD PRDLOD(xprulib1/prulod1)
MONMSG MSGID(CPF0000 SMU0000)
DLTPRDLOD PRDLOD(xprulib2/prulod2)
MONMSG MSGID(CPF0000 SMU0000)
DLTPRDLOD PRDLOD(xprutraining/prulodtrn)
MONMSG MSGID(CPF0000 SMU0000)
/**************************************************************************/
/* Create the Product Message File and add the message description */
/* (product description) */
/**************************************************************************/
START: CRTMSGF MSGF(XPRULIB/PRUMSGF) TEXT('Access.ability + package message file')
ADDMGD MSGID(MSG0001) MSGF(XPRULIB/PRUMSGF) + MSG('Access.Ability Option Base')
```
ADDMSGD MSGID(MSG0002) MSGF(XPRULIB1/PRUMSGF) +
MSG('Access.Ability Option 1 ')
ADDMSGD MSGID(MSG0003) MSGF(XPRULIB2/PRUMSGF) +
MSG('Access.Ability Option 2 ')
ADDMSGD MSGID(MSG0004) MSGF(XPRULIB3/PRUMSGF) +
MSG('Access.Ability Option 3 ')

/**************************************************************************/
/* */
/* Create Product Definition. */
/* *CODE load - 5001 */
/* */
/**************************************************************************/

CRTPRDDFN PRDDFN(XPRULIB/PRUPRODDFN) PRDID(1PRU001) +
RLS(V1R1M0) RGSID(*PHONE 5072866053) +
MSGF(PRUMSGF) PRDOPT((*BASE MSG0000 +
*ALWDYNNAM *NONE *CODEDF) (1 MSG0001) (2 +
MSG0002) (3 MSG0003 *ALWDYNNAM)) +
CPYRGTFST(*CURRENT) ALWMLTRLS(*YES)

/**************************************************************************/
/* *CODE Load OPTION *BASE */
/**************************************************************************/

CRTPRDLOD PRDLOD(PRULOD) PRDID(1PRU001) RLS(V1R1M0) +
OPTION(*BASE) LODTYPE(*CODE) +
LODID(*CODEDFT) RGSID(*PRDDFN) +
DVLLIB(*PRDDFN) PRILIB(*DVLLIB) +
ADLLIB((XPRUDTA) (XPRUSRV) (XDASLIB)) +
FLRL((XAAUPDT) ('XAAUPDT/COMMAND') +
('XAAUPDT/COMMAND/BIN') +
('XAAUPDT/COMMAND/DATA') +
('XAAUPDT/COMMAND/DLL') +
('XAAUPDT/TOOLBOX') +
('XAAUPDT/TOOLBOX/BIN') +
('XAAUPDT/TOOLBOX/DATA') +
('XAAUPDT/TOOLBOX/DLL') +
('XAAUPDT/PRULETRS') ('XAAUPDT/DESCRIBE') +
('XAAUPDT/DESCRIBE/MACROS'))

/**************************************************************************/
/* *CODE Load OPTION 1 */
/**************************************************************************/

CRTPRDLOD PRDLOD(PRULOD1) PRDID(1PRU001) RLS(V1R1M0) +
OPTION(1) LODTYPE(*CODE) LODID(*CODEDFT) +
RGSID(*PRDDFN) DVLLIB(xprulib1) +
FLRL((XPRUPHON))

/**************************************************************************/
/* *CODE Load OPTION 2 */
/**************************************************************************/

CRTPRDLOD PRDLOD(PRULOD2) PRDID(1PRU001) RLS(V1R1M0) +
OPTION(2) LODTYPE(*CODE) LODID(*CODEDFT) +
RGSID(*PRDDFN) DVLLIB(xprulib2) +
FLRL((XPRUHLP))

238 AS/400 Software Life Cycle Management
CRTPRDL0D PRDL0D(PRUL0DTRN) PRDID(1PRU001) RLS(V1R1M0) + 
  OPTION(3) LODTYPE(*CODE) LODID(*CODEDFT) + 
  RGSID(*PRDDFN) DVLLIB(XPRUTRAIN) + 
  PRILIB(*DVLLIB)

CHGPRDOBJD OBJ(XPRULIB/*ALL) OBJTYPE(*ALL) + 
  PRDID(1PRU001) RLS(V1R1MO) OPTION(*BASE) + 
  LODID(*CODEDFT)

CHKPRDOPT PRDID(1PRU001)

PKGPRDOPT PRDID(1PRU001) OPTION(*BASE) + 
  ALWAPICHG(*SAME) RLS(V1R1MO) + 
  REPACKAGE(*YES)
14.3.3 Distributing and Installing the Sample Application Client Code

**Important Note**

The Client Access/400 Administration Function as described in this section is used to download a new application to the user’s PC. This function is only available in the OS/2 16 bits and DOS Extended clients. To download a new application on other Client Access clients, you should use some other mechanism for the initial install. You should include a package file (*.PKG file) and the trigger file (QPTFIDX) to enable your PC application to be updated by the AS/400 PTF process.

After reviewing the Client Access/400 update and administration functions earlier in this chapter, we should be ready to understand the following scenario. Once
packaged, the product 1PRU001 is ready to be distributed to and installed on the remote systems. Once the product is installed, we want to download the client code to the user’s PC and use the Client Access/400 administration and update functions to manage the distribution from the server (source directory) to the client (target directory). Use the following steps:

1. Add a .PKG file to the application’s root folder:

   Before saving and distributing your package, you need to add a package file in order for the Client Access/400 administration and update functions to download the new client code.

   a. Create the file I:\XAAUPDT\PRU001.PKG using the OS/2 DIR command:

   
   ```
   I:\XAAUPDT dir /F/S > PRU001.PKG
   ```

   b. Edit PRU001.PKG to convert it to the Client Access/400 package file format.

   ![Figure 116. Example of Using an OS/2 Editor to Create a .PKG File](image)

   The final package file PRU001.PKG looks similar to this:

   ```
   PKGF 1PRU001 product
   MBRF XAAUPDT\COMMAND
   MBRF XAAUPDT\DESCRIBE
   MBRF XAAUPDT\PRULETRS
   MBRF XAAUPDT\TOOLBOX
   MBRF XAAUPDT\PRU001.PKG
   MBRF XAAUPDT\OPTFIDX
   MBRF XAAUPDT\PRPTF.PKG
   MBRF XAAUPDT\COMMAND\BIN
   MBRF XAAUPDT\COMMAND\DATA
   MBRF XAAUPDT\COMMAND\DLL
   MBRF XAAUPDT\COMMAND\BIN\CMDTGT.EXE
   MBRF XAAUPDT\COMMAND\BIN\CMHLP.HLP
   MBRF XAAUPDT\COMMAND\BIN\CONSERVE.BAT
   MBRF XAAUPDT\COMMAND\BIN\CONSERVE.ICO
   MBRF XAAUPDT\COMMAND\BIN\IBMMOUSE.CMD
   MBRF XAAUPDT\COMMAND\BIN\RTDEMO2.EXE
   MBRF XAAUPDT\COMMAND\DATA\ACCNOKEY.BMP
   MBRF XAAUPDT\COMMAND\DATA\AE01.DAT
   MBRF XAAUPDT\COMMAND\DATA\AGTASST1.NFO
   MBRF XAAUPDT\COMMAND\DATA\CLNDXA.NFO
   MBRF XAAUPDT\COMMAND\DATA\CLNDXL.NFO
   MBRF XAAUPDT\COMMAND\DATA\NDEMO.MRO
   ```
Note: We are also adding the QPTFIDX trigger file and the package file PRUPTF.PKG that is used by the PTF process discussed later in this chapter.

2. On each AS/400 system where you are installing your product, do the following:

Configure the Client Access users that are using the package as Client Access/400 administered users. Refer to Client Access/400 for OS/2 Setup or Client Access/400 for DOS Ext Memory Setup for details on how to use the Client Access/400 Administration function.

3. Add an ADDP entry in the administered user’s PCSINFO.ADM file:

```
I:\QIWSADM\USER\SUE type pcsinfo.adm
ADDP C:\XAAUPDT\PRU001.PKG
```

When this entry is processed by the update function, a new directory is added to the PC (C:\XAAUPDT) and the package file (PRU001.PKG) is added to that directory.

4. Add a UPDT entry in the administered user’s CONFIG.PCS file. Use the following sequence of displays:

a. Start Client Access Administration
b. Work with User Configurations
c. Change User Configuration (Enter, Enter)
d. General Options
e. Update Personal Computer Application  

f. Add Update Control Information  

g. Update Personal Computer Applications  

Fill in the parameters on the following displays:

```
Add update control information
More:  |

Specify the information below so files for the specified application are automatically updated each time Client Access is started.

Application Description
PRUDENTIAL PACKAGE

Source directory (master copy)
I:\XAAUPDT

Target directory (your copy)
C:\XAAUPDT

Update Option .................. > 1. Update files
                                        2. List files
```

```
Add update control information
More:  |

Specify the information below so files for the specified application are automatically updated each time Client Access is started.

Update subdirectories .............. > 1. Yes
                                    2. No

Update directly without using temporary directory ........ > 1. Yes
                                                        2. No
```

h. Save and Exit.

Our user’s CONFIG.PCS file now looks similar to this:

```
I:\QIWADM\USER\SUE type config.pcs
RTYP CMGR
ADMN I:\QIWADM\USER\SUE,C:\PCSOS2
LCLN P2618671
MCAO 128,,2
SFLR 1,1,,RCHASM01
RMTN RCHASM01
UPDT I:\QIWSDS2,C:\PCSOS2,S,,Client Access/400
    UPDT I:\XAAUPDT,C:\XAAUPDT,S,,PRUDENTIAL PACKAGE
```

If you prefer to skip the menus and take a short cut, edit each user’s I:\QIWADM\USER\user_name\CONFIG.PCS file and add the corresponding UPDT entry.

5. Authorize the user to the path in the AS/400 shared folder; it is recommended to mark all the folders on the shared folder with public authority (*EXCLUDE) so that the administrator manages security for the administered users.

```
ADDDLOAUT DLO(*ALL) FLR(XAAUPDT) USRAUT((SUE *USE))
```

The next time Sue starts Client Access, the following happens:
1. The administration function is processed, and Sue’s C:\PCSOS2\PCSINFO.ADM file is updated copying the source file I:\QIWSADM\USER\SUE\PCSINFO.ADM.

2. Because there is an ADDP entry in PCSINFO.ADM and the package target directory and package file do not exist in Sue’s PC, C:\XAAUPDT is created and PRU001.PKG is copied to it.

3. Sue’s C:\PCSOS2\CONFIG.PCS file is updated, copying the source file I:\QIWSADM\USER\SUE\CONFIG.PCS (with the new UPDT entry added).

4. I:\XAAUPDT and C:\XAAUPDT are compared and all the entries in the package file PRU001.PKG processed.

If you want to delete a package from an administered user’s PC, use the following steps:

1. Change the ADDP entry in I:\QIWSADM\USER\SUE\PCSINFO.ADM to DLTP:
   - DLTP C:\XAAUPDT\PRU001.PKG

2. Remove the user’s *USE authority to the folder.

3. Remove the corresponding UPDT entry for this application from I:\QIWSADM\USER\SUE\CONFIG.PCS. You may use the Client Access/400 Administration Function menus or edit the file and remove the entry.

   I:\QIWSADM\USER\SUE\CONFIG.PCS. now looks similar to this:

   ```
   I:\QIWSADM\USER\SUE type config.pcs
   RTYP CMGR
   ADMN I:\QIWSADM\USER\SUE,C:\PCSOS2
   LCLN P2618671
   MCAO 128,,2
   SFLR 1,,RCHASM01
   RMTN RCHASM01
   UPDT I:\QIWSOS2,C:\PCSOS2,S,,,Client Access/400
   ```

   The next time Sue runs STARTPCS, her C:\PCSOS2\PCSINFO.ADM and C:\PCSOS2\CONFIG.PCS files are updated and the PRU001 package is deleted; all the files in the MBRF entries are removed and PRU001.PKG is deleted. The directories and subdirectories remain in Sue’s PC.

14.3.4 Adding an Exit Program

In our scenario, let’s assume that after the files in the package file PRU001.PKG are installed, we want to create one additional directory and subdirectory where our client/server application will place some user files. You can run these exit or entry programs from the PC or from the AS/400 shared folder. Of course, the program must be packaged and distributed with your product the same way the other AS/400 exit programs are.

1. Add the Exit program to the root folder:

   Our simple program PRU001.CMD looks similar to this:
   ```
   - I:\XAAUPDT> type pru001.cmd
   ```

   ```
   REM Exit pgm for 1PRU001
   MD C:\XAAUPDT\PRUDAT
   MD C:\XAAUPDT\PRUDAT\CONTROL
   ```

2. Make sure the trigger file QPTFIDX is in the source root folder:
For a new installation, QPTFIDX does not exist on the target directory and the update function considers it a new package installation; therefore, it looks for the PCSUPDT.DAT file on the source directory to determine the name of the entry and exit programs.

3. For a new release installation, the VRM (Version Release Modification) values in QPTFIDX at the source and target directories are different:

Our scenario is a new package installation; QPTFIDX does exist on the source directory (shared folder) and does not exist on the target (PC). Our QPTFIDX file is:

```
VIRIMO 1PRU001
```

4. Add the PCSUPDT.DAT file in the application root folder (XAAUPDT) and specify the name of the Entry and Exit program (or programs). In our scenario:

```
- I:\XAAUPDT type pcsupdt.dat

REM Exit program name for 1PRU001
EXIT PRU001.CMD
```

---

14.4 Maintaining your Client/Server Product

In the previous sections of this chapter, we discussed how to package a client/server application into an AS/400 product, how to distribute the product to the server AS/400 systems, and presented some options that can be used to distribute client code from the AS/400 server folder to the users’ PCs. In the remaining sections, we explain how to use the PTF process discussed in previous chapters to service client/server products.

14.4.1 PTF Client Code

PTFs to change PC code are nothing else than PTFs to change the documents in the folders. Client code must follow the rules for folders on the AS/400 system listed in Section 14.1, “Packaging Structure of Client/Server Applications” on page 225. The concepts used in the PTF process for documents are similar to those explained when we discussed the PTF process for AS/400 objects.

On the PTF development system (Central Site), the fix for the document must be developed in a subfolder directly below the folder where the document to be fixed resides. We call this subfolder PTF development folder and its name must always be “/QP”. When the PTF is created using the CRTPTF command, the PTF function copies the document from a subfolder with the name specified in the PTFDOC keyword and “/QP” appended.

On the Service Requester (remote site system where the PTF is applied), the PTF document is installed during the apply PTF operation into the product’s folder; in other words, the PTF development folder “/QP” does not exist on the serviced system.

The Client Access/400 trigger file QPTFIDX must be created as a document in each root folder of your product. The contents of this document are used by the Client Access Update function during the PTF processing. This document triggers the process performed by the update program. Therefore, this
document should not be used by the product to store any information. For more information, refer to Section 14.2.1, “Client Access/400 Update Function, Trigger, and Package Files” on page 226 and Section 14.2.1.1, “Trigger Files” on page 227. Initially, before any PTF is applied, the content of QPTFDX in our scenario is:

```
V1R1M0 1PRU001
```

If your PTF is adding new PC files to your product or you want to delete existing PC files as part of the PTF process, you must use a package file (identified by the PKG extension) to add or remove PC files. Refer to Section 14.2.1.2, “Package File” on page 227 for more information on package files.

### 14.4.1.1 PTF to Change Existing Personal Computer Files

Getting back to our scenario, our product structure is discussed in Section 14.3.3, “Distributing and Installing the Sample Application Client Code” on page 240. We want to fix the file XAAUPDT\COMMAND\BIN\CONSERVE.BAT. The QPTFDX file was originally packaged and distributed with our product.

The following describes the tasks to be performed at the Service Provider system (Central Site) to develop a PTF to change the PC file CONSERVE.BAT:

1. Create the PTF development subfolder /QP in the corresponding path XAAUPDT\COMMAND\BIN.

   You can create a subdirectory from a PC working on the AS/400 shared folder:
   - `I:\XAAUPDT\COMMAND\BIN>md QP`

   You can obtain the same result by using the AS/400 command interface:
   ```
   Create Folder (CRTFLR)
   Type choices, press Enter.
   Folder . . . . . . . . . . . . > QP Name
   In folder . . . . . . . . . . > 'XAAUPDT/COMMAND/BIN'
   Text 'description' . . . . . . > 'PTF Development Folder'
   ```

2. Copy the PC file to be modified into the PTF development folder.

   If you are working on a PC using the AS/400 shared folder as your current drive:
   - `I:\XAAUPDT\COMMAND\BIN>copy conserve.bat qp\conserve.bat`
     1 file(s) copied.

   If you are using the AS/400 command interface:
   ```
   Copy Document (CPYDOC)
   Type choices, press Enter.
   From document . . . . . . . . > CONSERVE.BAT Name, *SYSOBJNAM
   From folder . . . . . . . . . . > 'xaaupdt/command/bin'
   To document . . . . . . . . . > *FROMDOC Name, *FROMDOC
   To folder . . . . . . . . . . . > 'xaaupdt/command/bin/qp'
   Replace document . . . . . . > *NO *NO, *YES
   ```
3. Using a PC editor, make the necessary modifications to the PC file XAAUPDT\COMMAND\BIN\QP\CONSERVE.BAT:
   - I:\XAAUPDT\COMMAND\BIN\QP>e conserve.bat

   ![CONSERVE.BAT code]

   Figure 117. Using an OS/2 Editor to Change PC File for PTF

4. Create the source member with your PTF cover letter. In our scenario, we created the PTF cover letter in PRUPTFLIB/QTXTSRC(CL1PR0001).

5. Create the PTF for the PC file CONSERVE.BAT:

   PGM

   START: CRTPTF PTF(1PR0001) LICPGM(1PRU0001) RLS(V1R1M0) +
   LODID(*CODEDFT) PTFOBJ (*NONE) +
   PTFDOC((CONSERVE.BAT +
   'xaaupdt/command/bin') +
   COVER((PRUPTFLIB/QTXTSRC CL1PR0001 2924)) +
   EXITPGM(*NONE) /* Create PTF 1PR0001 +
   - Fix for PC File CONSERVE.BAT - 1PRU001 */

   ENDPGM

   Let's look at the PTFs for the product 1PRU001:
   - WRKPTF

   ![Work with PTF (WRKPTF)]

   Work with PTF (WRKPTF)
   Type choices, press Enter.
   PTF identifier ............... *ALL Character value, *ALL...
   Product ...................... > 1PRU001 Character value, *SELEC
   Release level ................ *ONLY *ONLY, VxRxMx

Chapter 14. Packaging, Distributing, and Servicing Client/Server Applications 247
Now the PTF must be distributed to the service requester system (remote system) using SM/400. A PTF can be distributed from a service provider to a service requester through:

- The SNDPTF command
- The SNDPTFORD command
- Automatically searching the problem log for any symptom string associated with the PTF.

Once on the service requester system, the following sequence of steps takes place:

1. Load and apply the PTF, 1PR0001, in our scenario:

   ![Screen shot of Work with PTF system SYSNAM02]

   **Product ID . . . . . . :** 1PRU001  
   **Release . . . . . . . . :** V1R1M0  

   **Type options, press Enter. To work with assigned PTF IDs, press F18.**  
   *1=Create  3=Hold  4=Delete  5=Display details  6=Release  9=Work with problems  11=Load/Apply*  

   **Opt PTF Status PTF save file**  

   **1PR0001 Temporarily applied Yes (Release)**

2. The process of applying the PTF replaces the PC file (document) in the AS/400 folder (XAAUPDT/COMMAND/BIN/CONSERVE.BAT in our scenario):

   ![Screen shot of Work with Documents in Folders]

   **Folder . . . XAAUPDT/COMMAND/BIN**  
   **Position to . . . . . . Starting characters**  

   **Type options (and Document), press Enter.**  
   *3=Copy  4=Delete*  

   **Opt Document Document Description Revised Type**  

   - CMDTGT.EXE CMDTGT.EXE 11/30/93 PCFILE  
   - CMHLP.HLP CMHLP.HLP 11/30/93 PCFILE  
   - CONSERVE.BAT CONSERVE.BAT 02/21/94 PCFILE  
   - CONSERVE.ICO CONSERVE.ICO 11/30/93 PCFILE  
   - IBMMOUSE.CMD IBMMOUSE.CMD 11/30/93 PCFILE  
   - Q1PR0001.001 CONSERVE.BAT 11/30/93 PCFILE  
   - RTDEMO2.EXE RTDEMO2.EXE 12/02/93 PCFILE  

3. The Client Access/400 trigger file QPTFIDX is updated by the PTF process; the date and time stamp are changed and the PTF information is added:
4. The next time the update function runs, the time and date of the trigger file QPTFIDX is different in the source (AS/400 folder) and target (PC drive) directories. The update function assumes then, that the target directory is not current and all the files in the target directory are compared with the files in the source directory; if the date and time of the target and source files do not match, the file on the source directory is downloaded to the PC.

PTFs to PC Files can be temporarily applied, temporarily removed, permanently applied, and permanently removed the same as PTFs to any other AS/400 objects.

If a fix to your client/server product involves changes to the AS/400 objects as well as to the client code (PC Files), the PTF process allows you to synchronize those changes in one PTF that includes AS/400 library objects and documents (PC files), and when loaded and applied (or removed), guarantees that all the changes are performed in one operation.

14.4.1.2 PTF to Add New Personal Computer Files
As discussed in Section 14.2.1.1, "Trigger Files" on page 227 and Section 14.2.1.2, "Package File" on page 227, the trigger file QPTFIDX is used by the PTF process to update PC files that already reside in the target directory (PC). However, if you want to use the PTF process to add new files to your client code that were not initially part of your product, you must use a package file. Packages files are only processed by the Client Access update program when the package file exists in both the source and target directories and the time or date of the files in the two directories are different. There are multiple ways to handle the use of package files in the PTF process. We discuss one option here, but with the information provided in this chapter, you should be able to design the process that fits best in your environment.

We use the package file PRUPTF.PKG in the root folder in our PTF process to add new PC files. This file was part of our original package when the client code was installed (PRUPTF.PKG is a member of PRU001.PKG used in the initial installation).

The following list describes the tasks to be performed at the Service Provider system (Central Site) to develop a PTF to add the new PC file NEWFILE1.DAT:

1. Create the PTF development subfolder /QP in the corresponding path XAAUPDT\COMMAND\DATA.

2. You can create a subdirectory from a PC working on the AS/400 shared folder:
   - I:\XAAUPDT\COMMAND\DATA>md QP

3. You can obtain the same result by using the AS/400 command interface:
2. Create the new PC file that is added with the PTF in the PTF development folder:

- I:\XAAUPDT\COMMAND\DATA\QP>E NEWFILE1.DAT

3. PRUPTF.PKG is part of our 1PR0002 PTF as a PTF document (PTFDOC parameter). When the PTF is distributed to the service requester, PRUPTF.PKG is part of the PTF and, therefore, when 1PR0002 is loaded and applied, PRUPTF.PKG is updated.

To PTF the PRUPTF.PKG, we need to create the PTF development folder XAAUPDT\QP, copy PRUPTF.PKG to this subfolder, and add the corresponding MBRF entry:

4. Create the source member that contains the PTF cover letter. In our scenario, we created the PTF cover letter in PRUPTFLIB\QTXTSRC(CL1PR0002).

5. Create PTF 1PR0002 to add the new PC file NEWFILE1.DAT. Notice that we have included PRUPTF.PKG as a PTF document (parameter PTFDOC) and that the PTF development subfolder "/QP" is not specified; the PTF process expects the PTFDOC to be there and /QP is added to the folder path specified in the PTFDOC parameter.

```
PGM

START:  CRTPTF  PTF(1PR0002) LICPGM(1PRU001) RLS(V1R1M0) +
        LODID(*CODEDFT) PTFOBJ(*NONE) +
        PTFDOC('NEWFILE1.DAT +
        'XAAUPDT\COMMAND\DATA') (PRUPTF.PKG +
        XAAUPDT)) COVER((MLGPTFLIB\QTXTSRC +
        CL1PR0002 2924)) EXITPGM(*NONE) /* Create +
        PTF for 1PR0002 - Add new PC File +
        NEWFILE1.DAT */
```
Now the PTF must be distributed to the service requester system (remote system) using SM/400.

Once on the service requester system, the following sequence of steps takes place:

1. Load and apply the PTF, 1PR0002, in our scenario:

   ![Work with PTF](image)

   1. Type options, press Enter. To work with assigned PTF IDs, press F18.
   2. Opt PTF Status PTF save file
      - 1PR0002 Temporarily applied No
      - 1PR0001 Temporarily applied No

2. The process of applying the PTF adds the new PC file (document) to the AS/400 folder (XAAUPDT/COMMAND/DATA/NEWFILE1.DAT in our scenario) and updates the document XAAUPDT/PRUPTF.PKG:

   ![Work with Documents in Folders](image)

Note that PRUPTF.PKG and QPTFIDX have been updated by the PTF process on the service requester system.
The content of QPTFIDX has been updated:

- `J:XAAUPDT>type qptfidx`

```
V1R1M0 IPRU001 0000
IPRO001 XAAUPDT/COMMAND/BIN/CONSERVE.BAT
IPRO002 XAAUPDT/COMMAND/DATA/NEWFILE1.DAT
IPRO002 XAAUPDT/PRUPTF.PKG
```

3. The next time the update program runs, the time and date of the trigger file QPTFIDX is different in the source (AS/400 folder) and target (PC drive) directories. The update function assumes then, that the target directory is not current and all the files in the target directory are compared with the files in the source directory; if the date and time of the target and source files do not match, the file on the source directory is downloaded to the PC. In our scenario for PTF 1PR0002, no PC files have been updated but a new file added. Client Access update compares the date and time on the source and target package files PRUPTF.PKG. The PTF process has updated the package file in the source directory and, therefore, the new MBRF entries are processed and the file `C:XAAUPDT\COMMAND\DATA\NEWFILE1.DAT` is added to the PC.

Every time we want to add new PC files to our product client code, we update the file PRUPTF.PKG with new MBRF entries. Because every PTF that adds new PC files uses PRUPTF.PKG as one of its PTF documents (PTFDOC), a new PTF supersedes the previous one. For example, let’s create PTF 1PR0003 that adds `\XAAUPDT\COMMAND\DATA\NEWFILE2.DAT`. We follow the steps previously described but do not document every single one this time.

Before creating the PTF, we must update PRUPTF.PKG with the new MBRF entry:

- `I:XAAUPDT>type pruptf.pkg`

```
PKGF PRUPTF
MBRF COMMAND\DATA\NEWFILE1.DAT
MBRF COMMAND\DATA\NEWFILE2.DAT
REM Package file to add new PC files via PTF process
```

Create PTF 1PR0003:

```
PGM
START: CRTPTF PTF(1PR0003) LICPGM(1PRU001) RLS(V1R1MO) +
LODID(*CODEDFLT) PTFOBJ(*NONE) +
PTFDOC((NEWFILE2.DAT +
'XAAUPDT\COMMAND\DATA') (PRUPTF.PKG +
XAAUPDT)) COVER(*NONE) EXITPGM(*NONE) */+
Create PTF for 1PR0003 - Add new PC File +
NEWFILE2.DAT */
```

ENDPGM

After loading and applying PTF 1PR0003 at the service requester, PTF 1PR0002 is permanently applied (we specified *APYPERM in the Superseded PTF parameter in the LODPTF command).
PTF 1PR0002 is superseded by 1PR0003 because they have a PTF document in common (PRUPTF.PKG). The Display Superseded PTF option for PTF 1PR0003 shows the following:

On the AS/400 service requester, the following takes place by loading/applying PTF 1PR0003:

- The PTFDOC XAAUPDT/COMMAND/DATA/NEWFILE2.DAT is installed.
- The PTFDOC XAAUPDT/PRUPTF.PKG has replaced the previous version.
  - J:\XAAUPDT>type pruptf.pkg

```
PKGF PRUPTF
MBRF COMMAND\DATA\NEWFILE1.DAT
MBRF COMMAND\DATA\NEWFILE2.DAT
REM Package file to add new PC files via PTF process
```

- XAAUPDT\QPTFIDX has been updated:
  - J:\XAAUPDT>type qptfidx

```
VIRIMO 1PR0001 0000
1PR0001 XAAUPDT/COMMAND/BIN/CONSERVE.BAT
1PR0003 XAAUPDT/COMMAND/DATA/NEWFILE2.DAT
1PR0003 XAAUPDT/PRUPTF.PKG
1PR0003 XAAUPDT/COMMAND/DATA/NEWFILE1.DAT
```

Notice that there is no entry for the superseded PTF 1PR0002 and that the PTF document XAAUPDT/COMMAND/DATA/NEWFILE2.DAT now is part of PTF 1PR0003.

When the Client Access update program runs, the new file is added; PRUPTF.PKG and QPTFIDX are updated:
Comparing J:\XAAUPDT
with C:\XAAUPDT
Processing J:\XAAUPDT\PRUPTF.PKG
Adding C:\XAAUPDT\COMMAND\DATA\NEWFILE2.DAT
Updating C:\XAAUPDT\PRUPTF.PKG

Updating C:\XAAUPDT\QPTFIDX

PCSUPDT completed successfully

### 14.4.2 Summary

In this chapter, we discussed how to package, distribute, and service client/server applications.

With SM/400, you can package your AS/400 system and client code into a single unit as a product. You can distribute this product to remote AS/400 systems where, once installed, it is registered in the AS/400 software resource directory. SM/400 also allows you to service your client/server product using the IBM PTF process; fixes that involve changes to AS/400 system and client code can be combined in a single PTF, therefore, synchronizing the code changes in both platforms. AS/400 PTF support includes sophisticated change management features: temporarily apply or remove a PTF, permanently apply or remove a PTF, supersede a PTF, prerequisites for PTFs, PTF documentation, and so on. All of these features apply to PTFs that include AS/400 code, client code, or a combination of both. For a complete description of these features, refer to previous sections in this book.

*Client Access/400 Update and Administration* functions enable the administrator to centrally configure users’ files and control the distribution and update of software on the clients.
Appendix A. Diskette Install Instructions

There is a 3.5-inch diskette with this publication that contains the code for most examples quoted.

This code is provided for your use without restriction to use for demonstrations, educational use, or as a framework to build your own application. Please understand that the examples are not intended to be fully operational applications, but try to show certain aspects of packaging and using SystemView Managed System Services/400, Application Development Manager, and Application Dictionary Services.

The diskette holds a library called SG244187.1. Further instructions on installing and prerequisites follow.

Prerequisites: In order to be able to compile the restore programs and the examples in the library, the products required are:

• OS/400 5763-SS1 (V3R1, V3R2) or 5716-SS1 (V3R6, V3R7)
• ILE RPG 5763-RG1 (V3R1, V3R2) or 5716-RG1 (V3R6, V3R7)

To run the Application Development Manager scenarios, you need:

• ADTS/400 5763-PW1 (V3R1, V3R2) and feature 1613
  or
• ADTS/400 5716-PW1 (V3R6, V3R7) and feature 2213

To run the Application Dictionary Services scenarios, you need:

• ADTS/400 5763-PW1 (V3R1, V3R2) and feature 1612
  or
• ADTS/400 5716-PW1 (V3R6, V3R7) and feature 2212

To copy files from the diskette to a shared folder, you need Client Access/400 installed on your workstation.

Installation: Use the following steps:

1. Sign on as QSECOFR on a programmable workstation:
   A user profile with equal rights as QSECOFR works as well.
2. Create on the AS/400 system, a folder named SG244187 in the root directory.
3. Make the folder available as a shared folder for the programmable workstation.
4. Insert the diskette in drive A.
5. Copy all of the files from the diskette into shared folder SG244187.
6. Add a member to a source file (QCLSRC) in your own library:
   ADDPFM FILE(yourlib/QCLSRC) MBR(CLPROP) SRC👋
7. Use the CPYFRMPCD to copy the install program source from the folder:
   CPYFRMPCD FROMFLR(SG244187) TOFILE(yourlib/QCLSRC) +
   FROMDOC(CLPROP) +
   TOFILE(CLPROP)
As a result of the next action, you have a library installed called SG244187.1. Verify before running the program that a library of that name does not already exist on your system; otherwise, it is deleted.

8. Create and run this CL program.

9. Delete the following objects; they are of no further use:
   - Folder: SG244187
   - Source and program CLP41871 in yourlib.
Appendix B. Utility programs

This appendix describes some techniques that may be used to work with a PARTL such as:

1. Select a PARTL to work with.
2. Add a part to a (selected) PARTL.
3. Remove a part from a (selected) PARTL.
4. Use a PARTL to checkout all of the parts in it.

5. SOURCES:
   The sources for this solution are in the source file SG244187.1/QPARTLSRC.

Furthermore, a solution for browsing a source part in the Application Dictionary Services environment.

- Browse the related source of an object part.
- SOURCES:
  The sources for this solution are in the source file SG244187.1/QADSSSRC.

B.1 Making Changes to a PARTL File

The standard technique used by ADM/400 to add/change/delete entries within a PARTL file is to call a DFU utility program. This technique has several drawbacks. When the user is working with entries to be included in a PARTL, they may be working from a hard copy list of the desired parts or from a display of available parts.

If working from a hard copy, the user must correctly type the part name one entry at a time into the DFU data entry display. In addition to being awkward, there is no check that the names just entered truly represent valid part names.

If working from the PDM work with parts list, the user must correctly remember the part name and part type to enter this data into the PARTL after entering user option 2 for change.

By either working from a hard copy or working with a parts list, the user is not likely to enter the text or description without some automated process.

The following program, together with some user-defined options, makes the process of entering this data much simpler and with less chance of ending up with errors in the PARTL. With this program, the user adds or removes entries from a selected PARTL simply by entering a user-defined option beside the part to be added or removed from a part list.
Figure 118. DDS for the PARTL File

Figure 119 (Part 1 of 3). RPG Program to Add/Remove Entries to a PARTL File
Figure 119 (Part 2 of 3).  RPG Program to Add/Remove Entries to a PARTL File
Figure 119 (Part 3 of 3). RPG Program to Add/Remove Entries to a PARTL File

B.2 Installing the ADDPART Program

The RPG ADDPART program needs an external definition of a PARTL file to compile. To create this file, compile the DDS source MYPARTL and place the physical file in a library where it is available when compiling the RPG program ADDPART. The file MYPARTL is not used after the RPG program is compiled and may be later deleted. The ADDPART program must be located in a library that is available when the desired ADM project is being used. The ADDPART program must be placed in a library that is available to any user creating a part list. This can either be in an ADM group or a library available to everyone such as QGPL or QUSR0S.

To call the program, user-defined options may be added to invoke the program with the correct parameters. The correct parameters are passed from PDM to the program by the use of the user-defined options as shown in Figure 120 on page 261. In this sample, we have used the user options XS, XA, and XR to call the program, and XD has been added to call the display physical file member command with the correct parameters to display the selected PARTL file.
B.3 Using ADDPART Program to Update PARTL File

The ADDPART program uses the Local Data Area on the AS/400 system to save the selected part list name between invocations. If other applications use this LDA, the ADDPART program may interfere with their use of the LDA.

The user must first select a PARTL part that will be updated. To do this, place option XS beside the valid PARTL part.

User options XA and XR may then be placed beside the parts that are to be added or removed from the selected PARTL.

Figure 121 on page 262 shows a sample calling of the program. In this example, several parts are being added to the previously selected PARTL and one is being removed.
Work with Parts Using PDM

Project . . . . . . . MLG
Specified group . . . . LAMONT
Position to . . . . . . . . Position to type . . . .
Type options, press Enter.
2=Change 4=Delete 5=Display 14=Build
16=Run 28=Check out 29=Check in 30=Promote ...

Opt Part Type Language Group
XA MLGNAMD FILE DSPF PRODUCTION
XA MLGNAML FILE LF PRODUCTION
XA MLGREFP FILE PF PRODUCTION
XA SZRTVLIB CMD CMD IMPORT
XA USCRTUSCMD CMD CMD IMPORT
XR USLOBJCMD CMD CMD IMPORT
MLGOBJ PARTL *NONE PRODUCTION

More...

Parameters or command
=====
F3=Exit F4=Prompt F5=Refresh F6=Create
F9=Retrieve F10=Command entry F23=More options F24=More keys

Figure 121. User Options XA and XR

If the user-defined option XD is added, it may be used to invoke the DSPPFM command to display the current status of a PARTL.

B.4 Errors That May Occur While Running the ADDPART Program

Several feedback messages may be received when using this program.

- The ADDPART program may not be found. The error message is:
  **CPD0170** Program ADDPART in library *LIBL not found.

  This probably tells us the ADDPART program is located in a group that the user does not have access to. Move the program to a library that is used by the user.

- The user may not be authorized to update the selected PARTL file. The error message is:
  **CPF5134** I/O error was detected in MYPARTL (C G S D F).

  This error is caused by the user attempting to update a part list without the correct authority.

  You must have an administrator authorize you to the correct group.

- If you attempt to add the PARTL to itself, you receive:
  **DSPLY** Do not define this PARTL within itself

- If you attempt to add a part to a part list twice, you receive:
  **DSPLY** Type parttype part partname already in selected PARTL

- If you attempt to remove a part from a part list and it is not currently in the part list, you receive:
  **DSPLY** Type parttype part partname not in selected PARTL
• If you attempt to select a part list for updating but your selection is not a valid PARTL part, you receive:

\textbf{DSPLY} This part is not a PARTL part

• If you attempt to add parts to a part list but have not selected a part list yet, you receive:

\textbf{DSPLY} Have Not selected PARTL file

\section*{B.5 Check Out PARTL Parts}

This is an example of how you can check out parts that are in a part list. A user-defined option should be used to run this program.

As a suggestion, we used the option “XP”, but you can use any other option if you want. The interface should be:

\texttt{XP \ call sg244187.1/chkoutprt1 (&l &n &zt &zn &zp &zg)}
B.6 Browsing a Source Part

Since Application Dictionary Services does not provide the capability of browsing in a source related to an object, (Application Dictionary Services only works with objects) this user option solution was developed. It receives the object information, extracts the related source information, and starts the source editor.
in browse mode. You can install this program by defining a user option in the Application Dictionary Services environment.

**Note:** The default option file for the Application Dictionary Services environment is different from the default PDM environment. The option definition should be:

```
DS SG244187.1/BRWPARTADS PART(&ZN) LIB(&L) OBJTYPE(&ZA)
```

---

**Figure 123. Command to Use the Browse ADS Source Part Program**

```
/* This program will display a PART based on the assumption that */
/* the source part name and the Object partname are the same */
/* Since ADS only works with references based on Objects, will */
/* a conversion be done from the language type to the *OBJTYPE */
/* and then the Source information is retrieved from the OBJECT */
/* This program receives the already converted *OBJTYPE */
/* * * * E N D O F S O U R C E * * * *
```

---

**Figure 124. Program to the Browse ADS Source Part**

```
/* Errors in the command will be taken care off by MONMSG(CPF0000) */
/* * * * E N D O F S O U R C E * * * *
```
B.7 Create a CRTPTF Command String

Using the CRTPTF command requires patience because you have to type a lot of object names and types accurately if you have a reasonable PTF size. This utility program types all of the names for you based on the contents of your PTF library that holds all of the objects names for the PTF. Type the following command:

```
SG244187.1/CRTPTFCMD EXITPGM(QPZ1MP0021) PTFLIB(MLGPTFLIB)
```

Figure 125. Command to Create the CRTPTF Command
**Figure 126 (Part 1 of 2). Program to Create the CRTPTF Command**
Figure 126 (Part 2 of 2). Program to Create the CRTPTF Command

B.8 Display the LICPGM and PTF Information of an Object

When you use the DSPOBJD command, you are able to display the information regarding the LICPGM and PTF number. Depending on the type of object, it is on the second or third display. If you create this option, you can get to the information must faster. The option is:

```
PI CALL SG244187.1/RTVOBJPTFI PARM(&N &L &T)
```

```
5716PW1 V38CMD 950929
SOUCE FILE . . . . . . . SG244187.1/QSRCCHB
MEMBER . ........  RTVOBJPTFI
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
100 /*------------------------------------------------------------------*/ 02/18/96
200 /* This program can help you in displaying quickly to which */ 02/18/96
300 /* LICPGM this object belongs, and whether this is still the */ 02/18/96
400 /* original product object or if it was already replaced in a */ 02/18/96
500 /* PTF process. */ 02/18/96
600 /* */ 02/18/96
700 /*------------------------------------------------------------------*/ 02/18/96
800 PGM PARM(&OBJ &LIB &TYP) 02/13/96
900 DCL VAR(&OBJ) TYPE(*CHAR) LEN(10) 02/13/96
1000 DCL VAR(&LIB) TYPE(*CHAR) LEN(10) 02/13/96
1100 DCL VAR(&TYP) TYPE(*CHAR) LEN(10) 02/13/96
1200 DCL VAR(&LIC) TYPE(*CHAR) LEN(16) 02/13/96
1300 DCL VAR(&PTF) TYPE(*CHAR) LEN(10) 02/13/96
1400 DCL VAR(&APAR) TYPE(*CHAR) LEN(10) 02/13/96
1500 DCL VAR(&USR) TYPE(*CHAR) LEN(10) 02/13/96
1600 DCL VAR(&MSG) TYPE(*CHAR) LEN(100) 02/13/96
1700 /* Retrieve the workstation name */ 02/18/96
1800 RTVJOB A JOB(&USR) 02/13/96
1900 /* Retrieve the object information */ 02/18/96
2000 RTVOBJD OBJ(&LIB)OBJ(&OBJ) OBJTYPE(&TYP) LICPGM(&LIC) + 02/13/96
2100 PTF(&PTF) APAR(&APAR) 02/13/96
2200 CHGVAR VAR(&LIC) VALUE('Object ' CAT &LIB ' + 02/18/96
2300 + CAT &OBJ ' CAT + 02/18/96
2400 from LIC: ' CAT &LIC 7 ) CAT ' + 02/18/96
2500 'CAT %SST(ALIC B 8) CAT ' CAT + 02/18/96
2600 &PTF ' CAT + APAR: ' CAT &APAR) 02/18/96
2700 /* Send the Object LICPGM and PTF information thru a Break message */ 02/18/96
2800 SNDBRKMSG MSG(&MSG) 02/18/96
2900 02/13/96
3000 ENDPGM
```

Figure 127. Program to Retrieve the LICPGM and PTF Information

268 AS/400 Software Life Cycle Management
Appendix C. Special Notices

This publication is intended to help customers, business partners, IBM system engineering representatives, IBM software engineers and IBM development project managers to understand the use, positioning and product integration of the SM/400, Application Development Manager and Application Dictionary Services on the AS/400 system. The information in this publication is not intended as the specification of any programming interfaces that are provided by the SM/400, Application Development Manager and Application Dictionary Services. See the PUBLICATIONS section of the IBM Programming Announcement for Application Development Manager, Application Dictionary Services and SM/400 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (“vendor”) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other
operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

- Application System/400
- AS/400
- COBOL/400
- IBM
- OfficeVision
- Operating System/400
- OS/2
- OS/400
- RPG/400
- SystemView

The following terms are trademarks of other companies:

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other trademarks are trademarks of their respective companies.
Appendix D. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 273.

- AS/400 V3R1: Managing AS/400 Networks with Operations Control Center/400, SG24-4372-01.
- AS/400 V3R1: Managing PC Software with SM/400 and NetView DM/2, SG24-4371.
- Inside Client Access/400 for Windows 3.1 Version 1 Release 1 Modification 1, SG24-4429-02.

D.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. Order a subscription and receive updates 2-4 times a year at significant savings.

<table>
<thead>
<tr>
<th>CD-ROM Title</th>
<th>Subscription Number</th>
<th>Collection Kit Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>System/390 Redbooks Collection</td>
<td>SBOF-7201</td>
<td>SK2T-2177</td>
</tr>
<tr>
<td>Networking and Systems Management Redbooks Collection</td>
<td>SBOF-7370</td>
<td>SK2T-6022</td>
</tr>
<tr>
<td>Transaction Processing and Data Management Redbook</td>
<td>SBOF-7240</td>
<td>SK2T-8038</td>
</tr>
<tr>
<td>AS/400 Redbooks Collection</td>
<td>SBOF-7270</td>
<td>SK2T-2849</td>
</tr>
<tr>
<td>RISC System/6000 Redbooks Collection (HTML, BkMgr)</td>
<td>SBOF-7230</td>
<td>SK2T-8040</td>
</tr>
<tr>
<td>RISC System/6000 Redbooks Collection (PostScript)</td>
<td>SBOF-7205</td>
<td>SK2T-8041</td>
</tr>
<tr>
<td>Application Development Redbooks Collection</td>
<td>SBOF-7290</td>
<td>SK2T-8037</td>
</tr>
<tr>
<td>Personal Systems Redbooks Collection</td>
<td>SBOF-7250</td>
<td>SK2T-8042</td>
</tr>
</tbody>
</table>

D.3 Other Publications

These publications are also relevant as further information sources:

- System Manager Use, SC41-3321

- SAA AD/Cycle Application Development Manager/400 User’s Guide, SC09-1376-01

- SAA AD/Cycle Application Development Manager/400 Introduction and Planning Guide, GC09-1377-01

- Application System/400 Application Dictionary Services/400 User’s Guide, SC09-1492-00

- AS/400 System API Reference V3R6M0, SC41-4801

- AS/400 System API Reference V3R1M0, SC41-3801

- Application Development by Example, SC41-9852

- Client Access/400 for OS/2 Setup, SC41-3520

- Client Access/400 for DOS Ext Memory Setup, SC41-3500

© Copyright IBM Corp. 1996
- *Client Access/400 Optimized for OS/2 API and Technical Reference*, SC41-3511
How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**
  
  To get LIST3820s of redbooks, type one of the following commands:
  
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  
  To get lists of redbooks:
  
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  
  To register for information on workshops, residencies, and redbooks:
  
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  
  For a list of product area specialists in the ITSO:
  
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
- **Redbooks Home Page on the World Wide Web**
  
  http://w3.itso.ibm.com/redbooks
- **IBM Direct Publications Catalog on the World Wide Web**
  
  
  IBM employees may obtain LIST3820s of redbooks from this page.
- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**
  
  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.
How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

<table>
<thead>
<tr>
<th>In United States:</th>
<th>IBMMAIL</th>
<th>Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online Orders</td>
<td>usib6ipl at ibmmail</td>
<td><a href="mailto:usib6ipl@ibmmail.com">usib6ipl@ibmmail.com</a></td>
</tr>
<tr>
<td>Mail Orders</td>
<td>calibmbkz at ibmmail</td>
<td><a href="mailto:imannix@vnet.ibm.com">imannix@vnet.ibm.com</a></td>
</tr>
<tr>
<td>Fax</td>
<td>dkiibmsh at ibmmail</td>
<td><a href="mailto:bookshop@dk.ibm.com">bookshop@dk.ibm.com</a></td>
</tr>
</tbody>
</table>

- **Telephone orders**

  | United States (toll free) | 1-800-879-2755 |
  | Canada (toll free)        | 1-800-IBM-4YOU |
  | Outside North America    | (long distance charges apply) |
  | (+45) 4810-1320 - Danish  | (+45) 4810-1020 - German |
  | (+45) 4810-1420 - Dutch   | (+45) 4810-1620 - Italian |
  | (+45) 4810-1540 - English | (+45) 4810-1270 - Norwegian |
  | (+45) 4810-1670 - Finnish | (+45) 4810-1120 - Spanish |
  | (+45) 4810-1220 - French  | (+45) 4810-1170 - Swedish |

- **Mail Orders** — send orders to:

<table>
<thead>
<tr>
<th>IBM Publications</th>
<th>IBM Publications</th>
<th>IBM Direct Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publications Customer Support</td>
<td>144-4th Avenue, S.W.</td>
<td>Sortemosevej 21</td>
</tr>
<tr>
<td>P.O. Box 29570</td>
<td>Calgary, Alberta T2P 3N5</td>
<td>DK-3450 Allerød</td>
</tr>
<tr>
<td>Raleigh, NC 27626-0570</td>
<td>Canada</td>
<td>Denmark</td>
</tr>
</tbody>
</table>

- **Fax** — send orders to:

  | United States (toll free) | 1-800-445-9269 |
  | Canada                    | 1-403-267-4455 |
  | Outside North America    | (+45) 48 14 2207 (long distance charge) |

- **1-800-IBM-4FAX (United States) or (+1) 415 855 43 29 (Outside USA)** — ask for:

  - Index # 4421 Abstracts of new redbooks
  - Index # 4422 IBM redbooks
  - Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com

- **On the World Wide Web**

  - Redbooks Home Page http://www.redbooks.ibm.com

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).
IBM Redbook Order Form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Please put me on the mailing list for updated versions of the IBM Redbook Catalog.

First name                                  Last name

Company

Address

City                                      Postal code          Country

Telephone number                         Telefax number        VAT number

• Invoice to customer number

• Credit card number

Credit card expiration date          Card issued to          Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.
### Abbreviations

<table>
<thead>
<tr>
<th>ABBREVIATION</th>
<th>MEANING</th>
<th>RPG</th>
<th>SM/400</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM/400</td>
<td>Application Development Manager/400</td>
<td>ILE</td>
<td>SystemView System Manager/400</td>
</tr>
<tr>
<td>ADS/400</td>
<td>Application Dictionary Services/400</td>
<td></td>
<td>SystemView Managed System Services/400</td>
</tr>
<tr>
<td>AS/400</td>
<td>Application System/400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CISC</td>
<td>Complex Instruction Set Computer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CL</td>
<td>Control Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display File</td>
<td>DSPF</td>
<td>V1</td>
<td></td>
</tr>
<tr>
<td>DFU</td>
<td>Data File Utility</td>
<td>V2</td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td>Logical file</td>
<td>V1R1M0</td>
<td>Version 1 Release 1 Modification 0</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machine</td>
<td>V2R1M0</td>
<td>Version 2 Release 1 Modification 0</td>
</tr>
<tr>
<td>ITSC</td>
<td>International Technical Support Center</td>
<td>V2R3</td>
<td>Version 2 Release 3</td>
</tr>
<tr>
<td>MLG</td>
<td>Mailing List Applications</td>
<td>V3R0M5</td>
<td>Version 3 Release 0 Modification 5</td>
</tr>
<tr>
<td>MSGF</td>
<td>Message file</td>
<td>V3R1M0</td>
<td>Version 3 Release 1 Modification 0</td>
</tr>
<tr>
<td>PF</td>
<td>Physical file</td>
<td>V3R2M0</td>
<td>Version 3 Release 2 Modification 0</td>
</tr>
<tr>
<td>PGM</td>
<td>Program</td>
<td>V3R6M0</td>
<td>Version 3 Release 6 Modification 0</td>
</tr>
<tr>
<td>OS/2</td>
<td>Operating System/2</td>
<td>V3R7M0</td>
<td>Version 3 Release 7 Modification 0</td>
</tr>
<tr>
<td>OS/400</td>
<td>Operating System/400</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Index

Special Characters
/QP 245
*KEYED compliance 187
*PRDDFN 53
*PRDLOD 53

Numerics
1PRU001 241

A
abbreviations 277
acronyms 277
Add License Key Information (ADDLICKEY)
command 186
Add Product License Information (ADDPRLICIC)
command 146, 182
adding
license key information 186
product license information 146, 182
ADDLICKEY (Add License Key Information)
command 186
ADDPART 260, 261
ADDPRLICIC 183
ADDPRLICIC (Add Product License Information)
command 146, 182
ADDPRLICIC command
DFTUSGLMT parameter 183
GRACE parameter 183
LICTRM parameter 183
administered user 230
administered users 242
administration function 228, 232, 240, 241
CONFIG.PCS 230
model configuration 229
PCSINFO.ADM 230
STARTPCS 230
advantage of packaging 49
ALWDYNAM 59, 76
analyze problem 205, 209
ANZPRB 205, 209
APIs
QLICOBJD 60, 78
QLZAREQ 182
QLZARLS 182, 189
QPDLGEO 214, 221
QSZRECOV 82, 86
QSZRTVPR 60, 78, 116
QUSCRTUS 60, 78, 219
QUSLOBJ 60, 78
QUSRTVUS 60, 78
Application Development Manager
ADM/400 in the AS/400 Environment 172
Application Development Manager (continued)
base project hierarchy 24
Build and Test the NLS Parts 46
building the application 26
calculate and build the part 92
check out the part 92
checkout NLS part 42
create part in V2 group 140
Create the Message File 58
Create the Postoperation Exit Program 58
Develop a fix using ADM/400 89
define part in V2 group 141
Edit the PARTL Parts 61
export and package MLG version 2 140
export the PARTL part 96
export the PARTL parts, package the
application 63
importing the application 24
mailing list application and ADM/400 23
Maintenance Project Hierarchy 89
our NLS project hierarchy 40
product definition, product load part in
ADM/400 55
Promote Parts 47
PTF documentation 93
run and test the fix 97
storing text in database files 40
testing the fix 94
transition to NLS 42
version 2 project hierarchy 138
Application Development Manager/400
benefits of 21
binding directory 170
part relationships 170
service program 170
trigger relations 171
Application Dictionary Services
ADM/400 and ADS/400 172
After the Impact Analysis 181
Introduction 172
MLZIP field reference hierarchy 175
Our MLG Project Dictionary 173
Overview of the ADS/400 Functions 173
working with the MLZIP field 174
Application Dictionary Services/400 6
Apply Program Temporary Fix (APYPTF)
command 101
applying
program temporary fix 101
APYPTF (Apply Program Temporary Fix)
command 101
ARCHIVE parameter
on BLDPART command 96

© Copyright IBM Corp. 1996
attribute
  LOGRMTPRB 197

B
bibliography 271
binding directory 169
binding source 169
BLDPART 26
BRMS/400 11
build option 169

C
Call 33
Change License Information (CHGLICINF) command 187
Change Physical File (CHGPF) command 129, 131
changing
  license information 187
  physical file 129, 131
CHGLICINF (Change License Information) command 187
CHGPF (Change Physical File) command 129, 131
CHGPRDOBJD command 53
CHKMSGID 40
CISC 154
Client Access Administration 242
Client Access/400 242
  administered user 230
  administered users 242
  administration function 228, 232, 240
  package file 226, 227
  trigger file 226, 227
  update function 226, 232, 240
client/server application 225, 226
code load 53
command
  SNDPRD 66
command, CL (continued)
  SNDPTF (Send Program Temporary Fix) 101
  Work with License Information (WRKLICINF) 187
  WRKLICINF (Work with License Information) 187
compliance 181
concurrent 182
CONFIG.PCS 230, 231, 243
configuration files 229
constraints 130
CoOperative Development Environment/400 11
cover letter 93, 120
CPYPTFSAVF 196
CRTPART 34
CRTPRDFFN 59, 76
CRTPRJ 24
CRTPTF 97, 207, 245
CRTPTFPKG 196

D
definition of PARTL 27
Delete User Profile (DLTUSRPRF) command 161
deleting
  user profile 161
development folders 49
development libraries 49
DFTUSGLMT parameter 183
DFU 257
DLTPTF 196
DLTUSRPRF (Delete User Profile) command 161
DLTUSRPRF command
  OWNOBJOPT parameter 161
document control 10
dropping support
  for old release 150
DSPSFWRSC 53, 82
dynamic naming 59, 76, 81, 106, 137
dynamic naming consideration 106

E
document control 10
ENTRY program 232, 245
ERRMSG 39
ERRMSGID 40
example of exit program (PTF) 159
example of product exit program 161
exit program 103, 232, 245
exit program (PTF), example 159
exit programs 53, 114
exporting 36

F
feature 183
FFDC (first failure data capture) 212
first failure data capture 218, 223
  probes 218
tips and hints 223
first failure data capture (FFDC) 212, 213
detecting program 213
detection ID 212
point of failure 212
probe 212
QSFWERRLOG 212
reporting program 212
symptom string 213
fix for database file 105

G
GENLICKEY command 184
PRCGRP parameter 185
USGLMT parameter 185
VNDSTA parameter 185
GRACE parameter 183

H
HLDPTF 196

I
IBM OfficeVision/400 10
IBM Personal AS/2 Project Management (PAS/2) 8
ILE
  naming conventions 169
IMPPART 25
install program to add/change entries in PARTL 257
Installation
installing RCVPART 32

J
job
  QECS 202
  QESTP 202
journaling 130

K
keyed 182

L
language load 53
Library
  license information
    changing 187
    working with 187
  license key information
    adding 186
License Management
  license usage 182
license user handle 189
LICREQC 189, 190
LICRLSC 189, 191
LICTRM parameter 183
LMG2924 51
Load Definition
  Load Program Temporary Fix (LODPTF) command 101, 156
loading
  program temporary fix 101, 156
LODPTF (Load Program Temporary Fix) command 101, 156
LODTYPE(*CODE) 53
LODTYPE(*LNG) 53
log software error (see QPDLOGER) 214
LOGRMTPRB attribute 197

M
machine readable information 50
Machine Readable Instructions
Mailing List Application
  functional scenario 13
  inquire into mailing list master file 13
  Library structure 80
  Mailing List application description 13
  maintain mailing list master file 14
  packaging library structure 50
  parts structure 18
  query mailing list file 18
  submit mailing by account number 17
  submit special analysis report 17
Mark Object for Product Load
Message
  MLG2923 52
  MLG2924F 52
  MLGCODELIB 50
  MLGDATALIB 51
  MLGMIGPKG 83
  MLGMNUC 192
  MLGSRCLIB 51
  MLGV1PKGCA 73
  model configuration 229
  module 169
  MRI
    See machine readable information
  MSGCON 39
  MSGID 40
  multiple data libraries 136
  MYPARTL 260
program 169
Program Temporary Fix
  applying 101
  create a PTF 97
  Create PTF with exit program example 124
  example cover letter TXTSRC part 94
  export example 122
  loading 101, 156
  objects and location in creating PTF 97
  promote fix within ADM groups 121
  promote the part 95
  releasing 101
  sending 101
  test case 125
program temporary fix (see PTF) 87
Project Management
  Personal AS/2 Project Management Tool 9
  Project Control 9
  project planning 8
  Project Tracking 9
promote codes 90
PRU001.PKG 241
PRUPTF.PKG 242, 249
PTF 87, 102, 114, 115, 120, 123, 196, 245
  corequisite 196
  cover letter 87, 120
  exit program 102, 114
  prerequisite 87, 102, 196
  process overview 87
  process tips and hints 102
  superseded 87
  temporary objects 115, 123
PTF development folder 245, 246
PTF inventory 212
PTF notification 196
PTF process 249
PTFDOC 245

Q
QAPZCOVER 87, 98
QDNUPFST 178
QECS job 202
QESTP job 202
QLICOBJD 60, 78
QLZAREQ 182
QLZARLS 182, 189
QPDLOGER 214, 216, 221
QPLYBLDP 46
QPTFIDX 227, 232, 244, 245, 248, 249, 252
QPZ1MP0021 116, 123
QSRVDMP 223
QSZRECOV 82, 86
QSZRTVPR 60, 78, 116
QUSCRTUS 60, 78, 219
QUSLOBJ 60, 78
QUSRVTUS 60, 78
<table>
<thead>
<tr>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RCVPART</td>
<td>37</td>
</tr>
<tr>
<td>installing RCVPART command</td>
<td>37</td>
</tr>
<tr>
<td>registered</td>
<td>182</td>
</tr>
<tr>
<td>regression test</td>
<td>10</td>
</tr>
<tr>
<td>release license</td>
<td>189</td>
</tr>
<tr>
<td>Release Program Temporary Fix (RLSPTF) command</td>
<td>101</td>
</tr>
<tr>
<td>releasing</td>
<td></td>
</tr>
<tr>
<td>program temporary fix</td>
<td>101</td>
</tr>
<tr>
<td>REPLACE parameter on EXPPORT command</td>
<td>38</td>
</tr>
<tr>
<td>RISC 154</td>
<td></td>
</tr>
<tr>
<td>RLSPTF (Release Program Temporary Fix) command</td>
<td>101</td>
</tr>
<tr>
<td>root folder</td>
<td>225, 226</td>
</tr>
<tr>
<td>RSTLIC110</td>
<td>59, 60</td>
</tr>
<tr>
<td>RSTLIC11A</td>
<td>74, 76, 78</td>
</tr>
<tr>
<td>RSTLIC210</td>
<td>145</td>
</tr>
<tr>
<td>RSTLICPGM</td>
<td>53, 76</td>
</tr>
<tr>
<td>run the RCVPART command</td>
<td>37</td>
</tr>
</tbody>
</table>

| S                                                                 |                                                                 |
| Sample Programs                                                 |                                                                |
| ADDPART                                                        | 260, 261                                                      |
| DDS for the PARTL File                                          | 258                                                            |
| LICREQC                                                        | 189, 190                                                      |
| LICRLSC                                                        | 189, 191                                                      |
| MLGMMNUC                                                       | 192                                                            |
| MLGV1PKGCA                                                     | 73, 81, 83                                                    |
| OBGENERROR                                                     | 214                                                            |
| QPZ1MP0021                                                     | 116, 123                                                      |
| RPG Program to Add/Remove Entries to a PARTL File              | 258                                                            |
| RSTLIC110                                                      | 59, 60                                                         |
| RSTLIC11A                                                      | 74, 76, 78                                                    |
| RSTLIC210                                                      | 145                                                            |
| SZRTVLIB                                                       | 60, 78                                                         |
| SZRTVLIB1                                                      | 60, 78                                                         |
| SZRTVLIBPBT                                                    | 116                                                            |
| USCRRTFFDC                                                     | 218                                                            |
| USCRTUSCMCMD                                                   | 60, 78, 218                                                   |
| USCRTUSRPG                                                     | 60, 78, 218                                                   |
| USERFFDC                                                       | 219                                                            |
| USERERROR                                                      | 218                                                            |
| USLOBJRPG                                                      | 60, 78                                                         |
| USLOGJCMD                                                      | 60, 78                                                         |
| USOLOGICOBJ                                                   | 60, 78                                                         |
| USRTVUSRPG                                                     | 60, 78                                                         |
| SAVLICPGM                                                      | 53                                                             |
| SCHPTH part                                                    | 152                                                            |
| Send Program Temporary Fix (SNDPTF) command                    | 101                                                            |
| sending                                                        |                                                                |
| program temporary fix                                           | 101                                                            |

| SFLMSG                                                         | 39                                                             |
| SFLMSGID                                                       | 40                                                             |
| SM/400 release level                                            | 196                                                            |
| SNADS/ODF                                                      | 204                                                            |
| SNDPRD command                                                 | 66                                                             |
| SNDPTF (Send Program Temporary Fix) command                    | 101                                                            |
| SNDPTFORD                                                      | 203                                                            |
| SNDRVRQS                                                       | 203                                                            |
| SNDSRVRQS                                                      | 203                                                            |
| software estimation                                            | 11                                                             |

| Software Life Cycle                                           |                                                                |
| development process                                           | 3                                                              |
| IBM software tools positioning                                | 4                                                              |
| Management Elements                                           | 3                                                              |
| Managing the Backup and Recovery Process                      | 11                                                             |
| Managing the Document Control Process                         | 10                                                             |
| managing the testing process                                  | 10                                                             |
| Problem Management and Corrective Actions                     | 6                                                              |
| Project Control                                               | 9                                                              |
| project management                                            | 8                                                              |
| project planning                                              | 8                                                              |
| Project Tracking                                              | 9                                                              |
| Software Change Management                                    | 5                                                              |
| Software Life Cycle                                           | 6                                                              |
| software problem                                              | 214                                                            |
| source directory                                              | 227                                                            |
| STARTPCS                                                      | 230                                                            |
| STARTPCS.CMD                                                  | 230                                                            |
| STRADS                                                        | 173                                                            |
| superseded PTF                                                |                                                                |
| order of execution                                            | 103                                                            |
| symptom string                                                | 204                                                            |
| SYSTEMML part type                                            | 34                                                             |
| Systems Network Architecture Distribution Services             |                                                                |
| SystemView System Manager/400                                 |                                                                |
| Packaging task                                                | 71                                                             |
| SVSM/400 Product Packaging using ADM/400                      | 55                                                             |
| SZRTVLIB                                                      | 60, 78                                                         |
| SZRTVLIB1                                                     | 60, 78                                                         |
| SZRTVLIBPBT                                                   | 116                                                            |

| T                                                                 |                                                                 |
| target directory                                               | 227                                                            |
| temporary objects                                              | 123                                                            |
| test coverage                                                  | 10                                                             |
| TGTRLS                                                         | 153                                                            |
| threshold                                                      | 187                                                            |
| translation                                                    | 156                                                            |
| trigger file                                                   | 227                                                            |
| OPTFIDX                                                       | 227, 242                                                      |
| triggers                                                       | 130                                                            |
U
UPDPGM 165, 170
UPDSRVPGM 165, 170
USCRTFFDC 218
USCRTUSCMD 60, 78
USCRTUSRPG 60, 78
User Based Pricing
user options 261
user profile
deleting 161
USERFFDC 219
USGLMT parameter 185
using SNDPRD command 66
using the system list 36
USLOBJRPG 60, 78
USLOGJCMD 60, 78
USQLICOBJ 60, 78
USRTVUSRPG 60, 78

V
VNDDDTA parameter 185
VNDPWD password 184

W
warning 181
WITT 4, 10
Work with License Information (WRKLICINF) command 187
working with
license information 187
WRKLICINF (Work with License Information) command 187
WRKPRB 207
WRKPTF 207
WRKSPTPRD 195