Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

Second Edition (November 1996)

This edition applies to Version 3, Release 7, Modification Level 0, of Application Development Manager/400 (Feature 2213), a feature of IBM Application Development ToolSet/400 (Program 5716-PW1), and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change or addition.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 Eglinton Avenue East
North York, Ontario, Canada. M3C 1H7

You can also send your comments by facsimile (attention: RCF Coordinator), or you can send your comments electronically to IBM. See “Communicating Your Comments to IBM” for a description of the methods. This page immediately precedes the Readers’ Comment Form at the back of this publication.

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

Contents

Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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 independent created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Canada Ltd., Department 071, 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada. Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks and Service Marks

The following terms are trademarks of International Business Machines Corporation in the United States or other countries or both:

Application System/400	Operating System/400
AS/400	OS/2
C/400	OS/400
COBOL/400	PROFS
DB2	RPG/400
DB2 for OS/400	System/36
IBM	System/38
IBMLink	SystemView
Integrated Language Environment	VisualAge
Operating System/2	400

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.

About This Book

The Application Development Manager/400, a feature of the Application Development ToolSet/400 (ADTS/400) product, facilitates the development of application programs in an Application System/400 (AS/400) environment. Its activities fall into two categories: administration tasks and development tasks. This book contains information on:

- Functions of the Application Development Manager/400 feature
- Application Development Manager/400 part development commands
- Application Development Manager/400 administration commands
- How to work with search paths and user-defined types
- How to build, test, run, export, and distribute an application
- Application distribution to remote AS/400 systems
- Security considerations
- How to use the Programming Development Manager (PDM) utility
- How to manage System/36 applications
- System/38 considerations
- How to work with VRPG parts

Who Should Use This Book

This book is intended for project administrators who plan to create and manage projects defined for the Application Development Manager/400 feature, and for application developers who will be using this feature to develop applications.

Before using this book, you should be familiar with the following:

- General concepts about this feature. You should already have read the *ADTS/400: Application Development Manager Introduction and Planning Guide*.
- Your workstation (also known as a display station) and its controls.
- Your IBM AS/400 system and the software running on the Operating System/400 (OS/400) system.
- The Programming Development Manager (PDM) utility. See the *ADTS/400: Programming Development Manager* for information on this utility.

The Sample Project

This book uses a sample project based on a payroll application to illustrate the planning that needs to be done and ways to create a functioning application development environment. However, how the functions of the Application Development Manager/400 feature are implemented depends on an enterprise's particular needs and wants, its processes, and the nature of the application it plans to develop. The sample project, as it is used in the Application Development Manager/400 library, illustrates just one approach a development team might take.

Definitions

This book does not contain a glossary. New terms are defined where they first appear, and are highlighted in **boldface** type. For easy reference, each definition is also indexed under the term itself and under the index entry entitled *definitions*.

For More Information

You may also need to refer to other IBM manuals for more specific information about a particular topic. The *Publications Reference* describes all the manuals in the AS/400 library. Those that are most relevant to the Application Development Manager/400 feature are listed in the “Bibliography” on page 283.

The following online information is available:

- **Help for control language (CL) commands**

  There are two types of help available: **contextual** help and **extended** help. Contextual help explains a field. Extended help explains the purpose of a display or a command.

  To see the prompt display for a CL command, type the command, and press F4=Prompt. To see contextual help, type the command, and position the cursor on a field and press F1=Help. To see extended help, press F2=Extended Help when you are looking at the contextual help for that command.

  To see a list of all Application Development Manager/400 commands, type:

  ```
  GO CMDADM
  ```

  To see a list of project-related Application Development Manager/400 commands, type:

  ```
  GO CMDPRJ
  ```

  To see a list of group-related Application Development Manager/400 commands, type:

  ```
  GO CMDGRP
  ```

  To see a list of part-related Application Development Manager/400 commands, type:

  ```
  GO CMDPART
  ```

- **Help for Programming Development Manager displays**

  To see contextual help, position the cursor on a field, and press F1=Help. To see extended help, either position the cursor outside the areas for which contextual help is available and press F1=Help; or press F2=Extended Help while you are looking at contextual help.

Chapter 1. Introducing the Application Development Manager/400 Feature

The Application Development Manager/400 feature of the Application Development ToolSet/400 product provides application developers working in an AS/400 environment with a mechanism for efficiently and effectively managing application objects throughout the life of their applications. There are two types of users for this feature: project administrators and application developers. A **project administrator** is the person who defines a project hierarchy and enrolls users. An **application developer** is an application programmer who uses the Application Development Manager/400 feature to develop components of an application.

This chapter provides information on the following topics:

- Overview of project administration
- Overview of part development
- Application development environment
- Tools for application development

An Overview of Project Administration

For an organization or enterprise to use this feature, someone, in this case a project administrator, must create a project and its framework in the Application Development Manager/400 environment. The person who creates a project automatically becomes the administrator for that project with the necessary authority to create groups and enroll users in that project. A **group** is a collection of parts at the same phase in the development process. It is represented as an AS/400 library whose authority is controlled by the Application Development Manager/400 feature. A **project hierarchy** is a collection of groups organized into levels, with each level representing a phase in the development process.

The administrator establishes the framework of the application for others to work as he or she defines the project hierarchy. To create an effective hierarchy, the project administrator must understand the Application Development Manager/400 feature very well. In addition, this person must also have a great deal of knowledge about the information that will be under the control of this feature. The project administrator must also understand the processes the organization follows when developing and maintaining applications. Further, experience with the operating system and the system's utilities is essential.

Role of the Project Administrator

The scope of the project administrator's role and activities is determined by the application managed in the project hierarchy. A project administrator who uses this feature for the first time will typically perform the five steps illustrated in Figure 1 on page 2 to allow application developers to begin their work in the Application Development Manager/400 environment.

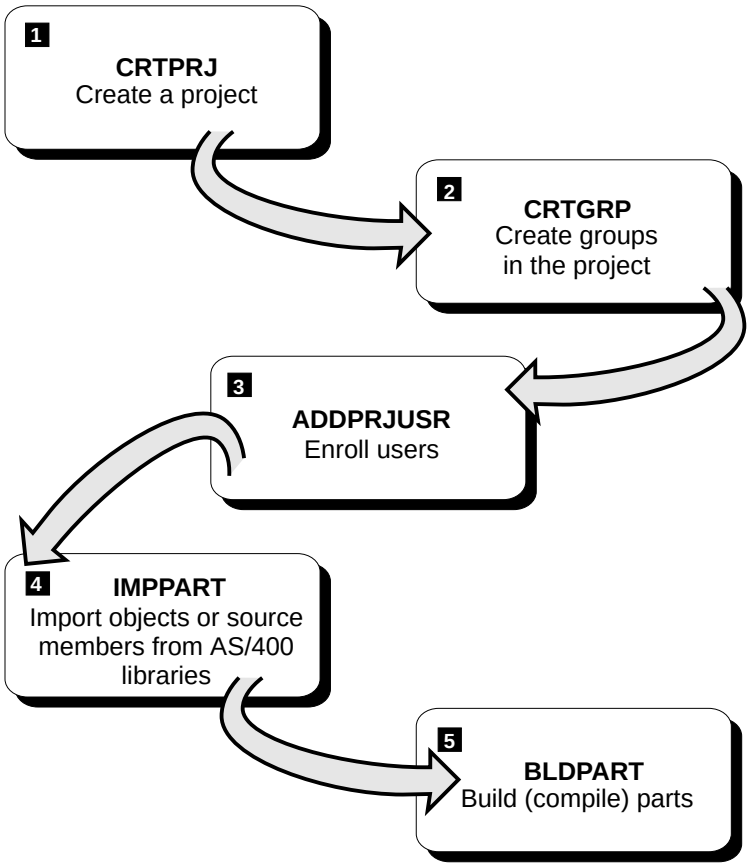

Figure 1. The Five Basic Steps Required to Begin Using this Feature

In addition, a project administrator's role may entail:

- Promoting elements of the project up the project hierarchy, for example, from a test level to a master level
- Testing the application
- Exporting the application from the Application Development Manager/400 environment hierarchy to a production or test environment
- Reclaiming project-hierarchy information after a system failure
- Developing parts or components of the application

Any AS/400 user can create a project. There is no special authorization required for the user profile of the person who is to act as the project administrator. The person who creates the project automatically becomes the project administrator for it. The authorized project administrator of one project does not automatically also have authorization to act as the project administrator of any other project. It is up to the project administrator of each project to enroll any co-workers as project administrators. The user profile QSECOFR has virtual enrollment as a project administrator in all projects. That is, QSECOFR is treated as project administrator in all projects regardless of whether QSECOFR is enrolled to the projects.

The role of project administrator is also distinct from that of the application developer. A project administrator has update access to everything in a project, whereas an application developer only has update access to specific work areas, or groups, as granted by the administrator. The basic administration tasks occur at the onset and completion of the application development activity, while development tasks take place throughout the development cycle. And, depending on the scope of the project and the size of the development team, the project administrator may also work as an application developer. Note too that a project administrator need not be the *system administrator*. In an AS/400 context, a system administrator is responsible for that which relates to the management of the entire OS/400 system.

An Overview of Part Development

Application development is the coding, compiling, running, debugging, and testing of application programs. An application developer can use the Application Development Manager/400 feature to do these tasks. This feature provides the framework in which to manage the development of the parts that make up application programs. A **part** is an object, 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 RPGSRC, RPGINC, PGM, or FILE, or you can define your own types using the Add ADM Type (ADDADMTYPE) 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 the ADDADMLANG command.

Before a developer can use the Application Development Manager/400 feature to perform part development tasks, a project hierarchy must exist. He or she must also be enrolled in the project and have update authority to a group. All of this is organized by the project administrator.

When a part needs to be updated, the developer must **check out** the part to a group to which they have update access. To check a part out, an appropriate version of it is copied into the developer's group, if necessary, and locked to prevent other developers from changing it. When the updates to the part are complete, the developer promotes the part out of his or her development group up to its parent group.

Multiple versions of a part can exist in a project at any given time. For example, when work on the first version of an application is complete, the project administrator can add groups to the project hierarchy so that developers can begin working on a second version of the application.

All tasks associated with creating, changing, or deleting Application Development Manager/400 projects, groups, and parts should be done using Application Development Manager/400 interfaces. These interfaces include CL commands, the Programming Development Manager displays, and the WRKPRJPDM, WRKGRPPDM, and WRKPARTPDM commands. If an administrator or developer changes or creates Application Development Manager/400 information using other interfaces, the results will be unpredictable. The build process may not recognize that the part has changed and may not recompile or process it. This leads to inconsistencies between parts and their corresponding AS/400 objects and members.

Parts can only be changed in the groups, or libraries, to which you have update access. If you decide to change a part outside the Application Development Manager/400 environment, the CHGPARTINF command updates the internal information for the part.

Role of the Application Developer

This feature allows you to develop versions of different parts in a controlled environment. In addition, more than one developer can update the parts of an application at the same time. This feature ensures that the changes made by one developer do not conflict with the changes made by another. The Building (compiling) of parts is automated.

The basic tasks an application developer does are:

- Creates new parts; copies parts either from within the control of this feature or by importing them from outside its control; and changes parts
- Builds parts using specific build scopes, build search paths, and build options
- Promotes parts up the project hierarchy
- Tests parts of an application within the Application Development Manager/400 environment or outside it by exporting parts to a test library

The Application Development Environment

Figure 2 on page 5 compares application development under the control of the Application Development Manager/400 feature and application development outside the control of this feature. The flow of boxes in the figure illustrates how you might typically create source code, edit it, and then compile it. The left side of the diagram shows the Application Development Manager/400 part development commands you use. The right side of the diagram shows the OS/400 commands you use if you are not developing parts under the control of this feature.

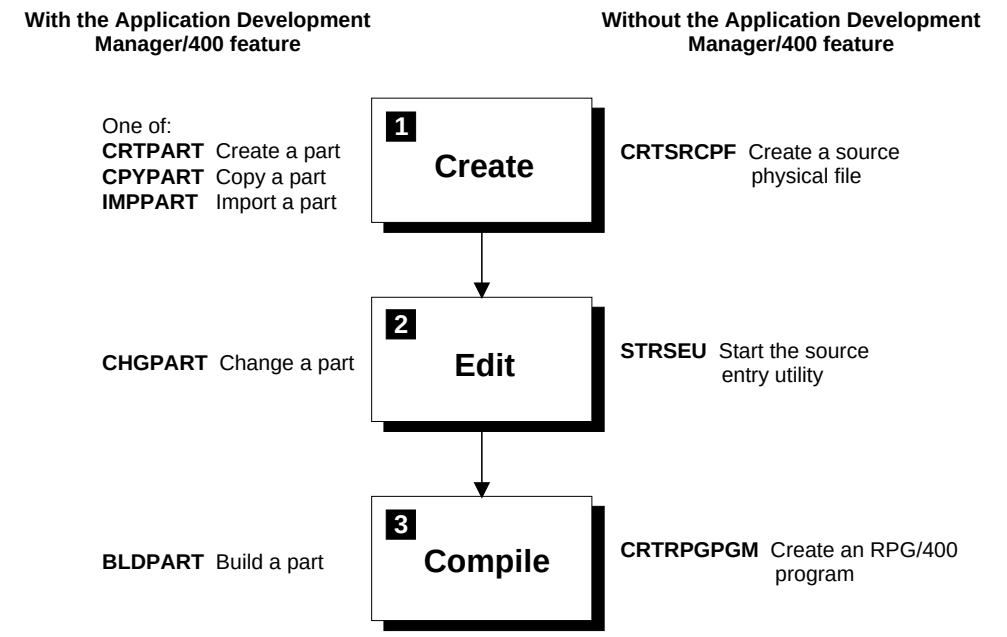

Figure 2. Application Development Environments

Tools for Application Development

The functions in the Application Development Manager/400 feature may be accessed using any one of the following interfaces:

- Control language (CL) commands from the AS/400 command line
- The Programming Development Manager (PDM) utility of the Application Development ToolSet/400 product
- The Application Dictionary Services feature of the Application Development ToolSet/400 product
- The Application Development ToolSet Client Server/400 (ADTS CS/400) product
- The IBM CoOperative Development Environment/400 (CODE/400) product
- The VisualAge for RPG product

.

Using the Application Development ToolSet/400 Utilities

Both administration tasks and development tasks can be performed using either the Programming Development Manager (PDM) utility or Application Development Manager/400 control language (CL) commands from the AS/400 command line. In this book, the first example of each command shows the CL command and its equivalent steps to follow using the Programming Development Manager utility. Appendix A, “Application Development Manager/400 Control Language Commands,” provides a complete list of these CL commands.

You can use the following PDM displays to perform these administration and development tasks:

Work with Projects Using PDM
Work with Groups Using PDM
Work with Parts Using PDM

For an overview of the PDM utility, see Chapter 16, “Using the Programming Development Manager Utility.”

There are several other Application Development ToolSet/400 utilities that are used with the Application Development Manager/400 feature. These include the Source Entry Utility (SEU), Screen Design Aid (SDA), Report Layout Utility (RLU), Data File Utility (DFU), File Compare and Merge Utility FCMU). You can simply access SEU, SDA, RLU and DFU from an option on the Work with Parts Using PDM display.

See the “Bibliography” for a list of Application Development ToolSet/400 publications on these utilities.

Using the Application Dictionary Services Feature

Application Dictionary Services is a feature of the Application Development ToolSet/400 product. The Application Dictionary Services feature is an impact analysis tool that you can use to assess the impact of a potential change to the application.

If you are using the Application Dictionary Services feature, you can also access the functions the Application Development Manager/400 feature offers to your application development environment.

You can access your Application Development Manager/400 parts in one of two ways:

1. Specify that you want to work with Application Development Manager/400 on the Start AppDict Services/400 (STRADS) display
2. Specify that you want to work with Application Development Manager/400 on the Work with Dictionaries display.

See the “Bibliography” for a list of Application Dictionary Services/400 publications.

Using the Application Development ToolSet Client Server/400 Product

The Application Development ToolSet Client Server/400 (ADTS CS/400) product consists of the following products:

- CODE/400
- VisualAge for RPG

See the “Bibliography” for a list of ADTS CS/400 publications.

Using the CODE/400 Product

You can use one of the CODE/400 editor, the CODE/400 Program Generator, or the WorkFrame interface, to access AS/400 libraries, files, and members or Application Development Manager/400 projects, groups, and parts using a graphical user interface.

The DDS Design Utility (DSU) component of CODE/400 allows you to open and build DDSSRC parts. As well, the Debug Tool component can be used to debug program parts.

See the “Bibliography” for a list of CODE/400 publications.

CODE/400 Editor: The CODE/400 editor allows you to select Application Development Manager/400 parts using a graphical user interface.

To access your Application Development Manager/400 parts:

1. Double-click on the CODE/400 editor icon.
2. Select *Open* from the *File* menu.
3. The Open window appears where you may select your Application Development Manager/400 parts.

Program Generator: The CODE/400 Program Generator allows you to build Application Development Manager/400 parts.

To build your Application Development Manager/400 parts:

1. Double-click on the CODE/400 Program Generator icon.
2. The Program Generator window appears where you may select your Application Development Manager/400 parts to be built.

Note: Refer to the *CODE/400 Self-Study Guide* for information regarding necessary changes required to be made to your BLDOPT files to enable CODE/400 error feedback and the Debug Tool.

WorkFrame: CODE/400 and WorkFrame make a powerful team. CODE/400 lets you edit, verify, compile, and debug your AS/400 applications using your workstation. WorkFrame lets you work with AS/400 objects on your desktop. It lets you manipulate AS/400 libraries, files, and members or Application Development Manager/400 projects, groups, and parts using an object-oriented user interface.

Using the VisualAge for RPG Product

VisualAge for RPG product offers a development environment for VRPG application programmers to develop, maintain, and document their visual applications on the workstation with AS/400 host interfaces.

The Application Development Manager/400 feature allows you to store and manage VisualAge for RPG applications in Application Development Manager/400 projects on the AS/400 system. For more information, see Chapter 17, “Working with the VisualAge for RPG Product” on page 239.

See the “Bibliography” for a list of VisualAge for RPG publications.

Chapter 2. Working with a Project

In the Application Development Manager/400 environment, an application is called a **project**. A project consists of all the related components of an application, regardless of the phase of development they are in. Thus, a project is not an AS/400 object. It is a set of AS/400 objects (parts) stored in AS/400 libraries (groups).

A project and the groups within it define the project hierarchy for application development. The Application Development Manager/400 feature uses the project hierarchy to determine the order of any library list needed by its commands. See Chapter 3, “Working with Groups in a Project Hierarchy” for more information on the project hierarchy and how parts in it are found by the Application Development Manager/400 commands.

This chapter provides information on the following topics:

- Creating a project
- Displaying a list of projects
- Changing a project
- Printing information about a project
- Deleting a project
- Working with a project using the Programming Development Manager utility

You should take the time to read the section in the *ADTS/400: Application Development Manager Introduction and Planning Guide* that discusses some of the things you should consider before you begin to create a project.

Creating a Project (CRTPRJ)

To use the Application Development Manager/400 feature, you must create a project. Use the Create Project (CRTPRJ) command to do this.

Any AS/400 user can create a project. Once you have created a project, you become the project administrator for it, and can use all the administration commands to work with it. You can add other project administrators. The user profile QSECOFR is an administrator for all projects.

Note

It is recommended that you create a separate Application Development Manager/400 project for each of your applications and another project for any parts common to several applications. The use of separate projects will improve the performance of the Work with Parts Using PDM display and will give you greater control of your applications.

When you create a project, you can select whether you want to have test data in physical files saved and restored automatically when you rebuild the physical file. The default for this parameter is SAVDTA(*YES). The build process saves the test data before building, and restores it after the physical file is rebuilt. If you specify SAVDTA(*NO), your data is deleted when the physical file is rebuilt.

When you create a project, you can provide a description of it on the TEXT parameter. For example, you will probably want to identify the name and purpose of the application in the text associated with the project.

Example: This example illustrates how to create a project called PAYROLL, with a short project name of PAY. The short project name can be up to 4 characters long and is used to create a unique library name for groups in a project. Any data associated with physical files is saved and restored when the build process occurs. As the TEXT parameter describes, the PAYROLL project is a weekly payroll processing application.

Using the CRTPRJ command

```
CRTPRJ PRJ(PAYROLL) SHORTPRJ(PAY) SAVDTA(*YES)
       TEXT('WEEKLY PAYROLL PROCESSING APPLICATION')
```

Using the Programming Development Manager utility

Press F6=Create on the Work with Projects Using PDM display.

Working with Projects

You can work with projects by typing CL commands directly on any command line. You can also use the commands in a CL program, a REXX procedure, or through QCMDEXC. The project commands are CRTPRJ, QRYPRJ, CHGPRJ, PRTPRJ, and DLTPRJ.

You can also work with projects through the Programming Development Manager utility using the Work with Projects using PDM display.

In addition to these project commands, this feature supports a Reclaim Project (RCLPRJ) command. This command resolves any inconsistencies in your project after a system failure or an abnormal end of a project administration command. See “Reclaiming a Project (RCLPRJ)” on page 204 for more information about this command.

Displaying All Projects (QRYPRJ)

Use the Query Project (QRYPRJ) command to obtain a list of all the projects to which you are enrolled on your AS/400 system. Only the projects that you are authorized to read or update appear on this list.

You could also use this command to verify that a project was created successfully, or to determine if any of the projects were damaged or made inconsistent by a system failure. See Chapter 15, “Securing Application Development Manager/400 Information” on page 201 for information about recovering projects.

Use the OUTPUT parameter to show where the output should go. The default spools the report for printing, OUTPUT(*PRINT).

Example: Figure 3 on page 11 illustrates what the spooled file looks like if you enter the following command:

```
QRYPRJ OUTPUT(*PRINT)
```

```
5716PW1   V3R7M0              Application Development Manager/400 - Query Project    11/08/96    11:59:48        Page . . :  0001

                               Short DBCS  Save
Project                        name  data  data  Text                                                          Damaged

PAYROLL                        PAY   No    Yes   WEEKLY PAYROLL PROCESSING APPLICATION                          No
           * * * * * E N D  O F  L I S T I N G  * * * * *
```

Figure 3. A Sample of a Spooled File Produced by the QRYPRJ Command

Alternatively, you can direct the output to an output file by specifying OUTPUT(*OUTFILE). The record format of the output file is the same as that used in the system-supplied database file QALYQPRJ in library QADM.

Example: The library list is used to determine where to store the output file called OUTFILE, and the first member in the file receives the output.

```
QRYPRJ OUTPUT(*OUTFILE) OUTFILE(*LIBL/OUTFILE) OUTMBR(*FIRST)
```

Changing a Project (CHGPRJ)

Use the Change Project (CHGPRJ) command to change the text associated with a project, or to change whether you want your test data saved and restored automatically when physical files are rebuilt in the project. Only project administrators can use this command.

If you specify SAVDTA(*YES), the build process saves the test data before building and restores it after the physical file is rebuilt.

Example: To change the descriptive text associated with the PAYROLL project, enter the following command:

Using the CHGPRJ command

```
CHGPRJ PRJ(PAYROLL) SAVDTA(*SAME) TEXT('BIWEEKLY PAYROLL
       PROCESSING APPLICATION')
```

Using the Programming Development Manager utility

Select option 2 (Change) on the Work with Projects Using PDM display.

This command will not change any of the enrollment information or change who is authorized to this project. To change enrollment information, use the CHGPRJUSR command, as discussed in the section “Changing User Enrollment Information (CHGPRJUSR)” on page 34.

Printing Information About a Project (PRTPRJ)

Use the Print Project (PRTPRJ) command to view the characteristics of a project and the groups within the project hierarchy. You could use this command after creating your project hierarchy to verify your promote path. Both project administrators and application developers can use this command.

You must specify the name of a project to which you are enrolled on the PRJ parameter of the PRTPRJ command.

The default value for the OUTPUT parameter, OUTPUT(*PRINT), spools the report for printing.

The report shows the PARTL required value, which indicates whether or not the group requires that a reason be given whenever a part is changed, created, or promoted. The value of Y indicates that a reason is required.

Example: This examples illustrates how to create the sample report shown in Figure 4.

Using the PRTPRJ command

```
PRTPRJ PRJ(PAYROLL) OUTPUT(*PRINT)
```

Using the Programming Development Manager utility

Select option 6 (Print) from the Work with Projects Using PDM display.

```
5716PW1   V3R7M0              Application Development Manager/400 - Print Project   11/08/96    13:37:40         Page . . :  0001

Project . . . . . . . . . . . . . . . :   PAYROLL
Short project name  . . . . . . . . . :   PAY
DBCS data . . . . . . . . . . . . . . :   No
Save physical file data . . . . . . . :   Yes
Text  . . . . . . . . . . . . . . . . :   WEEKLY PAYROLL PROCESSING APPLICATION
                                                Short                                                    PARTL
Level  Group                                    name   Promote code                  Notify      Req  Text
 01    MASTER                                   MAST   MASTER                       *NONE        N   MASTER group (root) for P
                                                                                                      AYROLL project.
 02    TEST                                     TST    MASTER                       *NONE        N   TEST group of the PAYROLL
                                                                                                       project
 03    DEVELOPER1                               DEV1   MASTER                       *NONE        N   DEVELOPER1 group of the P
                                                                                                      AYROLL project.
 03    DEVELOPER2                               DEV1   MASTER                       *NONE        N   DEVELOPER2 group of the P
                                                                                                      AYROLL project.
              * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 4. A Sample of a Spooled File Produced by the PRTPRJ Command

The level indicator for each group helps you determine the project hierarchy. The purpose of this level indicator is to clarify the hierarchical nature of the groups within the project. The project hierarchy described in this report is represented in Figure 5.

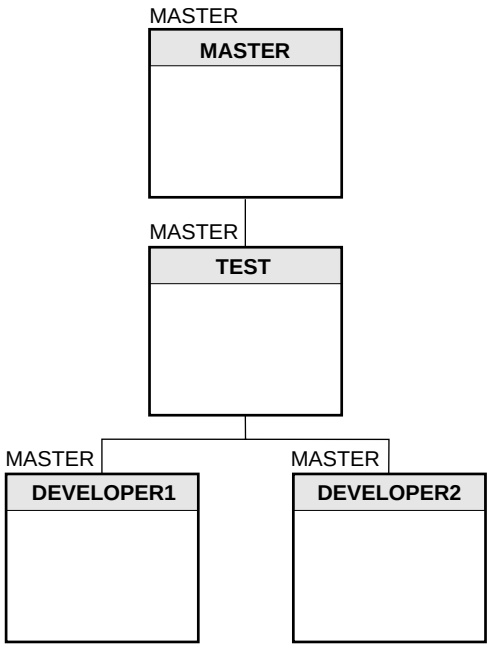

Figure 5. The Project Hierarchy Described in the PRTPRJ Report

Figure 5 and the sample report both illustrate a project with four groups and three levels to its hierarchy. The word MASTER that is listed in the report and appears outside and above each box represents the promote code defined for these groups.

Alternatively, you can direct the output to an output file by specifying OUTPUT(*OUTFILE). The record format of the output file is the same as that used in the system-supplied database file QALYPPRJ in library QADM.

Example: The library list is used to determine where to store the output file called OUTFILE, and the first member in the file receives the output.

```
PRTPRJ PRJ(PAYROLL) OUTPUT(*OUTFILE) OUTFILE(*LIBL/OUTFILE)
       OUTMBR(*FIRST)
```

Deleting a Project (DLTPRJ)

Use the Delete Project (DLTPRJ) command to delete an entire project from the Application Development Manager/400 feature. When you delete a project, all the groups within the project and all the parts within the groups are deleted, even if developers are working in those groups. Only project administrators can delete projects.

You must specify a project name on the PRJ parameter.

Note

When a project is deleted, the archive libraries as well as the group libraries are deleted. A library created outside of the Application Development Manager/400 feature is also deleted if its name is a valid archive library name for one of the group in the project. The archive libraries and the group libraries are deleted only if the owner is QPRJOWN, otherwise, a warning is issued. See “Archiving a Part” on page 94 for a description of valid archive library names.

Example: This example illustrates how to remove the project PAYROLL from the AS/400 system.

Using the DLTPRJ command

```
DLTPRJ PRJ(PAYROLL)
```

Using the Programming Development Manager utility

Select option 4 (Delete) on the Work with Projects Using PDM display.

The Programming Development Manager utility provides you with a confirmation display when you use option 4. You are not given a warning if you use this command directly from the command line.

Chapter 3. Working with Groups in a Project Hierarchy

Typically, a project consists of more than one group. Each group within the project represents a **version** or level of code in your application.

For example, a group called MASTER could represent the most current or up-to-date version of your entire application. Another group, TEST, could represent a mixture of changed and unchanged code that is still being tested. This group is created below the group MASTER in the project hierarchy, as shown in Figure 6.

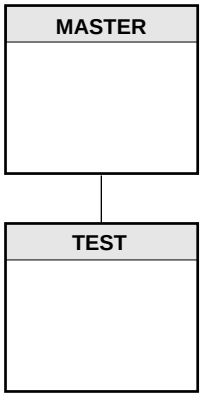

Figure 6. A Project Hierarchy with Two Groups

The project hierarchy identifies a search path. A **search path** is an arrangement of groups that determines the order in which the Application Development Manager/400 feature searches the various groups when looking for parts in a project hierarchy. When working in the group TEST, parts found within that group are used first because they represent the most recently changed versions of the parts in the application. Any parts of the application that are not found in the group TEST are looked for in the group MASTER. For more information about search paths, see Chapter 9, “Using Search Paths.”

This chapter provides information about the following topics:

- Creating a group
- Creating the project hierarchy
- Changing a group
- Deleting a group
- Working with groups using the Programming Development Manager utility
- Sharing a development group
- Enrolling developers and administrators to projects and groups
- Working with project user commands
- Creating a sample project hierarchy

You should take the time to read the section in the *ADTS/400: Application Development Manager Introduction and Planning Guide* that lists some of the things you should consider before you begin to define groups in a project.

Creating a Group (CRTGRP)

Use the Create Group (CRTGRP) command to create the groups for a project. You can only do this after the project has been created, and you must be a project administrator for the project to use this command. See “Creating a Project (CRTPRJ)” on page 9 for more information.

When you create a group, you must provide two names for it: a name for the group that is used in all group-related CL commands, and a short, five-character name. This short name is combined with the short project name to create a unique AS/400 name for each group in the project. See Appendix C, “Naming Rules” for information on how to name a group.

You must specify a value on the PARENT parameter to establish the relationship between the groups in a project hierarchy. When you are creating the first group in the project hierarchy, you must specify the special value *NONE because the group has no parent group. A **parent group** is the group immediately above another group in a branch of the project hierarchy. You should not change the promote code value when creating a project hierarchy for the first time. The default value is *PARENT. If you are creating a second or subsequent group in a project hierarchy, you must specify the name of the parent group under which this group is being placed.

A **coded character set identifier** (CCSID) can be associated with the group on the CCSID parameter. This identifier determines the character set identifier for the parts stored within the group. The default CCSID value, *PARENT, uses the CCSID associated with the parent group. If you are creating the first group in a project hierarchy, the CCSID associated with the job is taken as the default value.

You can indicate that you want the CCSID to be the same as the process or job running the CRTGRP command by specifying *JOB, or that you do not want data in source parts to be converted by specifying *HEX; or you can enter an integer value between 1 and 65535 to identify a specific CCSID. See “Understanding the CCSID” on page 275 for information on how the CCSID is used in applications supporting more than one national language.

Use the NOTIFY parameter to notify people that a part which exists in one branch of a project hierarchy is being checked out with the CHKOUTPART command to a group in another branch of the same project hierarchy. Refer to “Setting the Notification for a Group” on page 24 for more information about this parameter.

You can change the PARTLREQ parameter for a group. The PARTLREQ parameter specifies whether or not a PARTL name is required when parts are changed in this group. When *YES is specified for a specific group, a reason must be given when part development commands are used with this group as the target group.

When you create a group, you can provide a description on the TEXT parameter of what stage in the project the group represents.

Project Groups and the AS/400 Libraries List

The AS/400 system limits the user portion of the library list to 25 libraries. Each project-group pair in the project hierarchy that is part of a search path represents one library. The number of groups in each search path in the project hierarchy must not exceed 25 because of the AS/400 library list restriction. It is unlikely, however, that you will require more than 10 groups in a search path. See Appendix C, “Naming Rules” for information on how project and group names are combined to form unique AS/400 library names.

Creating the Project Hierarchy

Groups are further categorized depending on where they are located in the project hierarchy. The **root group** is the first, or top, group in any project hierarchy. You can only have one root group for each project hierarchy. Typically, it is the group that contains the complete, updated, and tested first version of your application. This is usually the group into which you import existing applications as well.

Example: This example illustrates how to create the group MASTER in the project PAYROLL. Figure 7 represents the hierarchy that is created as a result. *NONE is specified on the PARENT parameter to indicate to the Application Development Manager/400 feature that MASTER is a root group.

Using the CRTGRP command

```
CRTGRP PRJ(PAYROLL) GRP(MASTER) SHORTGRP(MST) PARENT(*NONE)
       TEXT('MASTER GROUP IN PROJECT PAYROLL')
```

Using the Programming Development Manager utility

Press F6=Create from the Work with Groups Using PDM display.

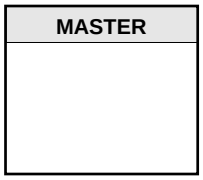

Figure 7. The Project PAYROLL with the Group MASTER

To create the second level in the project hierarchy, specify MASTER on the PARENT parameter for the next group.

Example: The following command creates the group TEST in the project PAYROLL, as illustrated in Figure 8 on page 18.

```
CRTGRP PRJ(PAYROLL) GRP(TEST) SHORTGRP(TST) PARENT(MASTER)
       TEXT('TEST GROUP IN PROJECT PAYROLL')
```

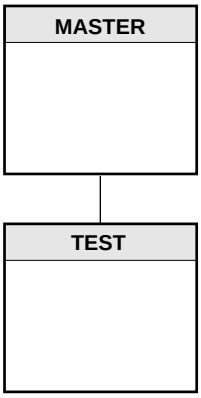

Figure 8. The Project PAYROLL with Two Groups

Now that the project hierarchy has the groups MASTER and TEST, you need to define some groups where developers can perform part-development tasks. A **development group** is a group in the project hierarchy where part development is done.

Development activities, such as creating parts, checking parts in or out, and changing or promoting parts, should ideally be done in the bottom or lowest groups in the project hierarchy. Developers should not be given update access to groups such as TEST and MASTER. Only after parts are changed and tested, should they be promoted up the hierarchy, to be tested again together with parts promoted from other groups, to make sure all the components work together correctly. Thus parts higher in the project hierarchy are more stable than those in groups lower in the project hierarchy. An additional consideration is that if changes were made by a developer to a copy of a part in an intermediate group in the project hierarchy, such changes could be replaced if another developer promotes the same part from a lower group. Changes of this kind should only be done if absolutely necessary and, if the changes are to be kept, the copy of the same part lower in the hierarchy should also be changed accordingly.

You may want to assign names to the development groups that clearly identify them as such. These names could be the user profile name of the person working in the group, or a more generic name such as DEVELOPER1 or DEVELOPER2.

Example: The following commands create two development groups in the project PAYROLL where developers can do part-development activities.

```
CRTGRP PRJ(PAYROLL) GRP(DEVELOPER1) SHORTGRP(DEV1) PARENT(TEST)
       TEXT('DEVELOPER1 GROUP IN PROJECT PAYROLL')

CRTGRP PRJ(PAYROLL) GRP(DEVELOPER2) SHORTGRP(DEV2) PARENT(TEST)
       TEXT('DEVELOPER2 GROUP IN PROJECT PAYROLL')
```

Figure 9 on page 19 represents the project PAYROLL as it appears with four groups.

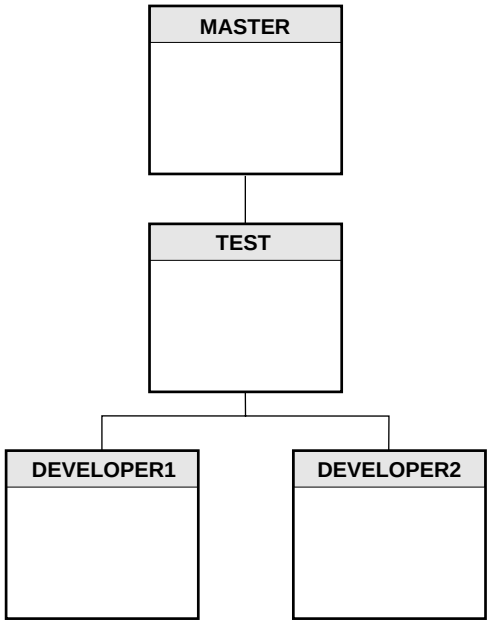

Figure 9. The Payroll Project with Four Groups

If you want to ensure that your application is built or ready to be exported to the TEST group, you can create a group between the TEST group and the development group, for example, to contain the parts for which development work is complete. You could call this group INTEGRATION. The purpose of this group is to preserve test versions of the application while allowing developers to check parts out from, or promote parts into, this group.

Defining One Promote Code for the Project Hierarchy

As parts are developed in the project hierarchy, they are promoted one group at a time, from the lower groups to the higher ones along a promote path. A **promote path** is the arrangement of groups between a group containing parts that must be promoted and the group that will eventually contain the parts when work on them is finished.

A promote path through a project hierarchy is determined by the promote code you specify on the PRMCODE parameter when you create a group. A **promote code** identifies the group to which a part can be promoted. This group is called the **target group**. The promote path controls how far up the project hierarchy a part can be promoted.

When you create a root group and do not specify a value on the PRMCODE parameter, the promote code for the group defaults to the group name. The Application Development Manager/400 feature determines whether the group you are creating is a root group by the value on the PARENT parameter.

When you create groups that are not root groups, the default value for the PRMCODE parameter causes the new group to have the same promote code as that for the group specified on the PARENT parameter.

One promote path for the project PAYROLL is created by accepting the default values for the PRMCODE parameter of the CRTGRP command. In this example, parts worked on in the development groups can eventually be promoted to the group MASTER, one level at a time. You must have update access to the group from which you are promoting.

You can specify the name of the promote code you want associated with the group on the CRTGRP command. See Appendix C, “Naming Rules” for information on how to name promote codes. In the next example, either of the following commands creates the group TEST with a promote code of MASTER.

Example: In the first example, the default promote code is used while in the second example, the promote code is specified. Because the promote code for the parent group is MASTER, the promote code for the TEST group is MASTER too.

```
CRTGRP PRJ(PAYROLL) GRP(TEST) SHORTGRP(TST) PARENT(MASTER)
       TEXT('TEST GROUP IN PROJECT PAYROLL')

CRTGRP PRJ(PAYROLL) GRP(TEST) SHORTGRP(TST) PARENT(MASTER)
       PRMCODE(MASTER) TEXT('TEST GROUP IN PROJECT PAYROLL')
```

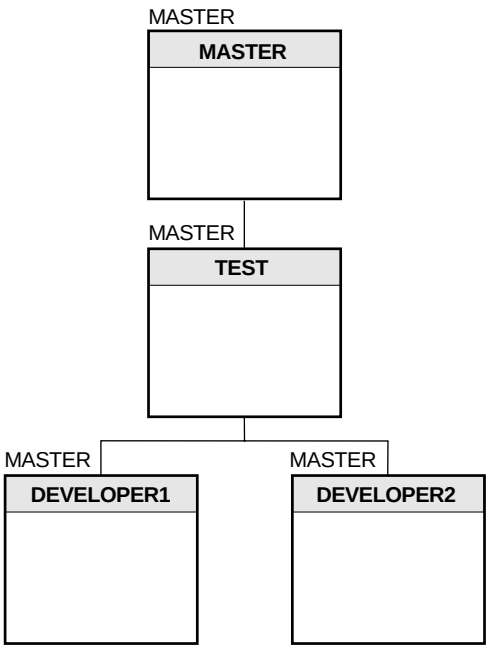

Figure 10. The Payroll Project with the Promote Code MASTER

One promote code for a project is effective if you have only one version of your application in the project hierarchy, or if you are starting new development activity for an application. If your organization must maintain an existing version of an application while updating another version of the same application, you may need several promote codes within your project hierarchy.

Structuring a Project Hierarchy

A project hierarchy can change over time to accommodate development work on several versions of an application. For example, when work on the first version of an application is complete, development of a second version of the application can begin. As well, development work on the first version may be needed to maintain the application. Figure 11 shows such a project hierarchy.

Parts in the V1 branch of the project hierarchy are promoted to group V1MASTER when work on the first version is complete. The V1FIX group is created for fixes that may have to be made to parts in the group V1MASTER.

Development of the second version of the application is ready to begin. The project administrator adds a new branch to the project hierarchy consisting of the groups V2MASTER, V2TEST, DEVELOPER3, and DEVELOPER4, all with the promote code V2.

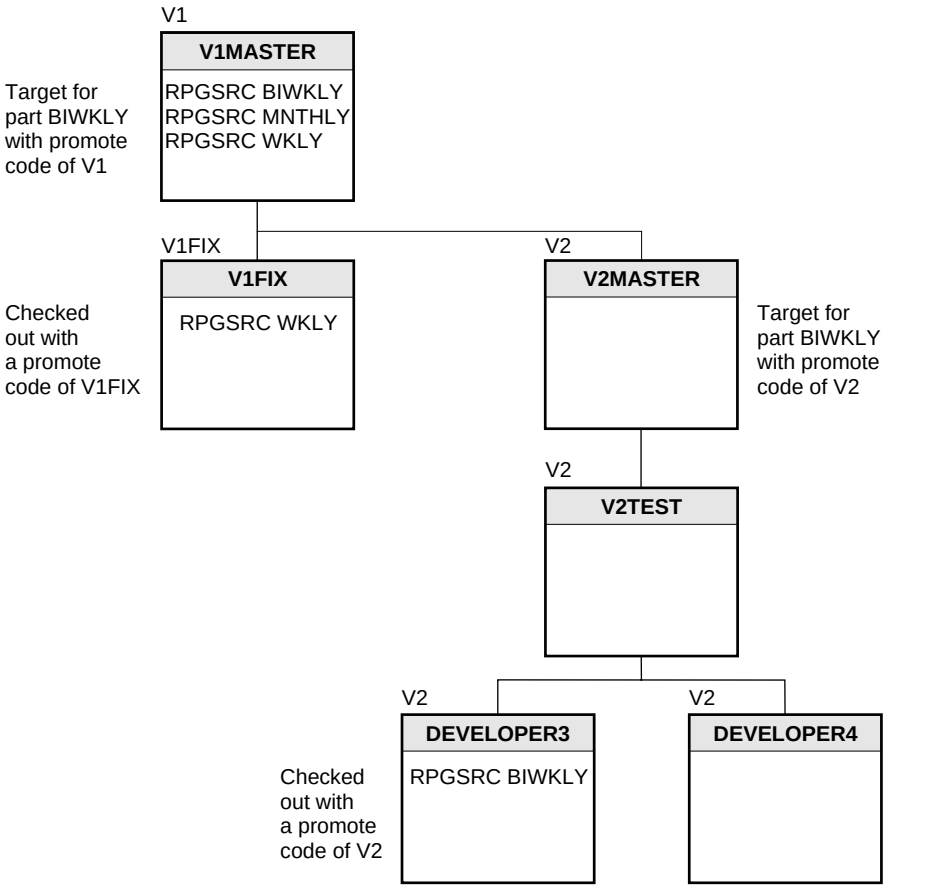

Figure 11. The Payroll Project with Three Promote Codes

Promote Codes

Several promote codes within one project hierarchy are defined in the same way as one promote code. You use the PRMCODE parameter on the CRTGRP command to define the promote code for the group. The combination of promote codes for the groups in the project hierarchy form the promote paths.

Example: The following CL commands were used over time to create a project hierarchy with six groups and three promote codes as shown in Figure 11 on page 21. In this example, assume that the project has already been created using the CRTPRJ command. This project hierarchy assumes that work on the first version of an application is complete, and that all parts have been promoted to the group V1MASTER. This hierarchy preserves the first version of the application while allowing fixes to be made to this version by checking parts out that need to be changed into the V1FIX group. It also allows changes to be made to the second version of the application.

```
CRTGRP PRJ(PAYROLL) GRP(V1MASTER) SHORTGRP(V1MST) PARENT(*NONE)
       TEXT('VERSION 1 GROUP IN PROJECT PAYROLL')

CRTGRP PRJ(PAYROLL) GRP(V1FIX) SHORTGRP(V1F) PARENT(V1MASTER) PRMCODE(V1FIX)
       TEXT('VERSION 1 FIX GROUP IN PROJECT PAYROLL')

CRTGRP PRJ(PAYROLL) GRP(V2MASTER) SHORTGRP(V2MST) PARENT(V1MASTER) PRMCODE(V2)
       TEXT('VERSION 2 GROUP IN PROJECT PAYROLL')

CRTGRP PRJ(PAYROLL) GRP(V2TEST) SHORTGRP(V2T) PARENT(V2MASTER)
       TEXT('VERSION 2 TEST GROUP IN PROJECT PAYROLL')

CRTGRP PRJ(PAYROLL) GRP(DEVELOPER3) SHORTGRP(DEV3) PARENT(V2TEST)
       TEXT('DEVELOPMENT 4 GROUP FOR VERSION 2')

CRTGRP PRJ(PAYROLL) GRP(DEVELOPER4) SHORTGRP(DEV4) PARENT(V2TEST)
       TEXT('DEVELOPMENT 5 GROUP FOR VERSION 2')
```

There are three promote paths: one that uses the V1 promote code, a second that uses the V1FIX promote code, and a third that uses the V2 promote code.

Each group in a project hierarchy can have only one promote code associated with it. The project can have many promote codes. However, when you create a group, the promote code you assign to it must not disrupt an existing promote path for any parts lower in the project hierarchy that have a target group above the new group you are creating. For example, in Figure 11 on page 21 you could not create a group with a promote code of V1 between the V2TEST group and the lower groups if there were parts in those groups with a promote code of V2. Doing so would disrupt the V2 promote path for these parts.

As in Figure 10 on page 20, the search path used in Figure 11 on page 21 to locate parts is determined by the group a developer is working in. For example, if a developer is working in the group DEVELOPER3, parts are searched for first in that group. If the part is not found there, the group V2TEST is searched. If the part is not found there, the group V2MASTER is searched. If the part is not found there, the group V1MASTER is searched. With this search path, the developer working in the group DEVELOPER3 does not *see* parts in the groups DEVELOPER4 and V1FIX.

Two Versions of One Part

In Figure 11 on page 21, there are three parts in the payroll project. One of the parts, BIWKLY, has two versions because new functionality is being added to it. The three parts are BIWKLY, MNTHLY, and WKLY. Suppose the first version of the payroll application is complete; however, errors have been discovered in these three parts that require some additional development activity. The parts BIWKLY, MNTHLY, and WKLY are checked out to the group V1FIX with a promote code of V1FIX.

At the same time, a developer working on the next version of the application determines that some new functionality was needed in the part BIWKLY. This developer checks BIWKLY out to the group DEVELOPER3 with a promote code of V2. Note that BIWKLY cannot be promoted from the group V2MASTER to the group V1 because the promote codes are not the same.

When the developer working in the group DEVELOPER3 is finished working with BIWKLY, the part is promoted to the group V2TEST. The administrator then promotes the part to the group V2MASTER after ensuring that it functions as expected.

Now when the second version of the payroll application is built, the version of BIWKLY in the group V2MASTER is used instead of the version in the group V1MASTER. Parts MNTHLY and WKLY did not change in the second version, so the copies of these parts found in the group V1MASTER are used.

Creating a Group without a Promote Code

You can also create a group with a promote code of *NONE. This means that the group does not have a promote code associated with it. Parts created or copied to a group with a promote code of *NONE cannot be promoted.

A group with PRMCODE(*NONE) could be created for a person assigned to test the application. This tester could then still check a part out or copy a part, change the part and test it, and check the part back in, but could not promote the changed part back up the project hierarchy.

Setting the Notification for a Group

When you set the NOTIFY parameter on for a group such as V1MASTER, notification messages are issued if a part that is already checked out to V1FIX, for example, is currently being worked on in the V2 branch of the hierarchy. If the part is changed, the change may have to be made to more than one version of the part. For example, when you create a group, you can use the NOTIFY parameter to advise users that a part that exists in one branch of a project hierarchy is being checked out with the CHKOUTPART command to a group in another branch of the same project hierarchy. If the part is changed, the other version of the part may require the same change.

When the value *DEVELOPER or a particular name is specified, any parts checked out from the group specified on the GRP parameter of the CHKOUTPART command cause notification to occur as described above. Notification messages are issued for all versions of a specific part and type that have a promote code different from the part being checked out and that exist in groups in other branches of the project hierarchy. If the existing part has a promote code of *NONE, notification occurs regardless of the promote code used to check the part out.

Notification messages are not issued for copies of the same part that exist in the group from which or to which the part is being checked out, or if the part is the output of the build process, for example, a part of type PGM. Typically a project administrator uses the default NOTIFY(*NONE), which means that no notification should occur.

When all the development work is complete for the first version of an application, and all the parts are in the group V1MASTER, the project administrator can change the NOTIFY parameter on the CHGGRP command for the group V1MASTER to specify who should be notified when the parts from that group are checked out. See Figure 12 on page 25. This helps to monitor the parts that are checked out for the next version of the application, and the parts checked out for fixes to the current version of the application.

When a part is checked out from the group V1MASTER, one of the following types of notification occurs:

- If the value *DEVELOPER is specified, the developer who has the part checked out in another branch of the project hierarchy receives a notification message. The developer who checked the part out receives the same message.
- If a user profile name is specified, that user profile is notified. The developer who checked the part out receives the same message. Anyone who has the part checked out in another branch of the project hierarchy also receives a message.

Consider the following diagram which illustrates three versions of an application. Development of the first version of the application is complete, and all parts are in the group V1MASTER.

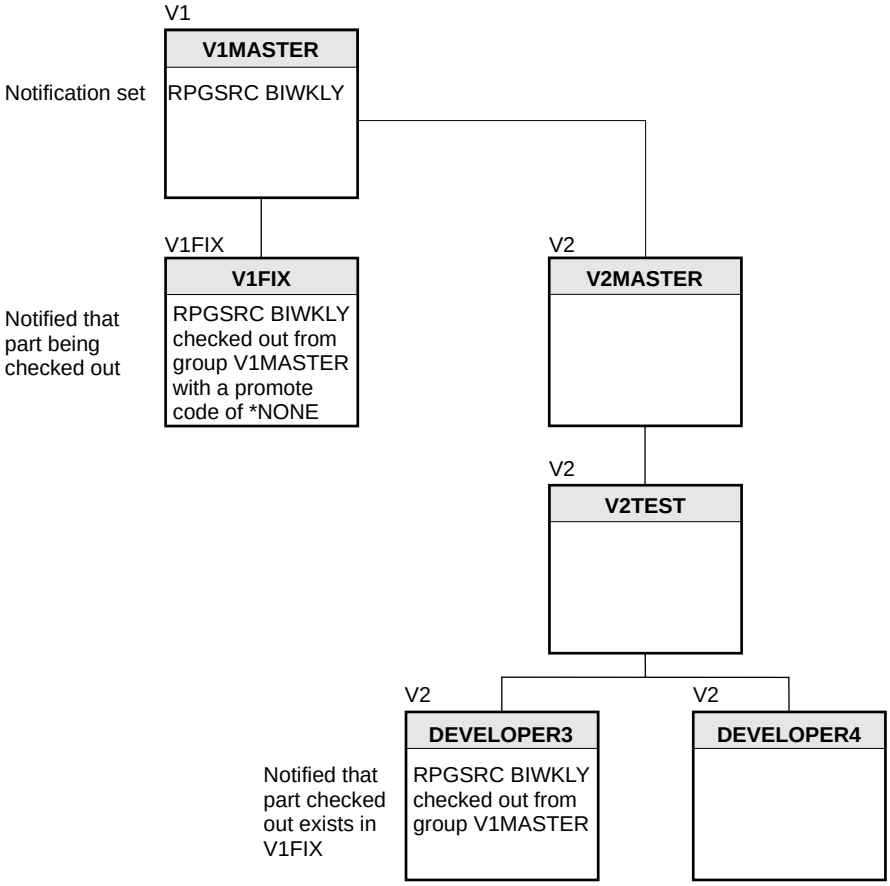

Figure 12. An Example of Notification

1. The project administrator uses the CHGGRP command and sets the NOTIFY parameter to *DEVELOPER for the group V1MASTER. Any part that is checked out from this group now causes a notification message to be sent. The developer who is checking the part out receives the same message.
2. A developer in the group V1FIX checks the part RPGSRC BIWKLY out from the group V1MASTER with a promote code of *NONE.

   Notification does not occur because the only copies of the part RPGSRC BIWKLY that exist are in the group the part was checked out from (V1MASTER) and the group the part was checked out to (V1FIX).
3. A developer in the group DEVELOPER3 checks the part RPGSRC BIWKLY out from the group V1MASTER. The part now exists in three groups: V1MASTER, V1FIX, and DEVELOPER3.

4. The developer who has the part checked out in the group V1FIX receives a notification message to indicate that the part is being checked out to the group DEVELOPER3. The developer who is checking the part out receives a notification message to indicate that the part has already been checked out to the group V1FIX.
5. The developer working on a fix when finished with it can then inform the developer who checked out the part in the group DEVELOPER3. That developer may use the CMPPART command to compare two versions of the parts in DEVELOPER3 and V1FIX groups. If required, the developer can then merge the fix using the MRGPART command.

If the part exists only in the group it was checked out from, or the group it is checked out to, a notification message is not issued.

Working with Groups

You can work with groups by typing CL commands directly on any command line. You can also use the commands in a CL program, a REXX procedure, or through QCMDEXC. The group commands are CRTGRP, CHGGRP, and DLTGRP.

In addition to these group commands, you can use the PRTPRJ command to display information about the groups in the project. See “Printing Information About a Project (PRTPRJ)” on page 11 for information on the PRTPRJ command.

You can also work with groups through the Programming Development Manager utility using the Work with Groups Using PDM display.

Changing a Group (CHGGRP)

Use the Change Group (CHGGRP) command to change the attributes of a group. Only project administrators can do this.

You might use this command to change the text associated with a group. You can also use it to change the parent of a group. Changing the parent allows you to rearrange your project hierarchy or insert a new group in the project hierarchy.

You can change the promote code associated with a particular group. The new promote code that you assign to the group must not disrupt an existing promote path for parts in groups below this one.

You can also change the notification that will occur when a part is checked out from a group. You can choose one of the following values on the NOTIFY parameter: *SAME, *NONE, *DEVELOPER, or the name of a user profile. *SAME is the default, and means that the current notification setting has been kept. See “Setting the Notification for a Group” on page 24 for more information.

The PARTLREQ parameter specifies whether or not a PARTL name is required when parts are changed in this group. When *YES is specified for a specific group, a reason must be given when part development commands are used with this group as the target group.

Changing the Groups in a Project Hierarchy

Figure 13 illustrates how a project hierarchy changes when a new group is added.

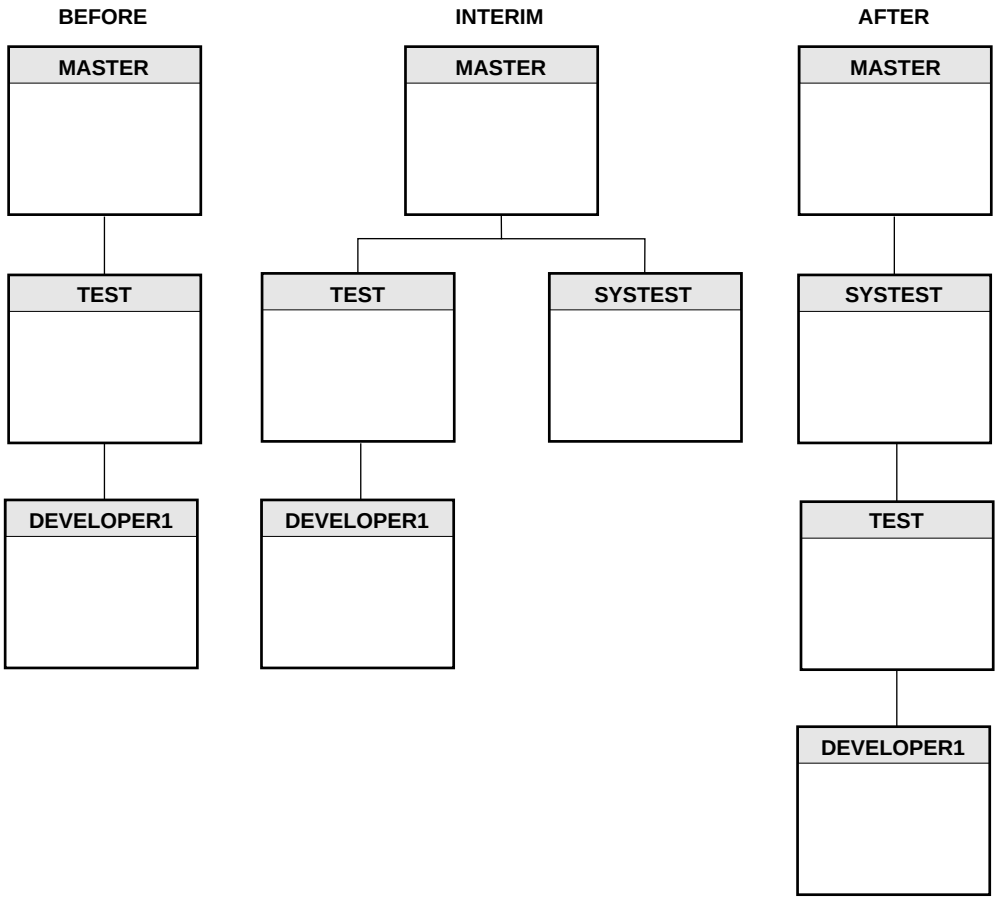

Figure 13. Changing a Project Hierarchy

The Before project hierarchy has three groups. Assume you want to change the project hierarchy so that there is a new system-test group between the group MASTER and the existing group TEST. The first step is to create the new group.

Example: The following CL command adds a new group and creates the Interim hierarchy.

```
CRTGRP PRJ(PAYROLL) GRP(SYSTEST) SHORTGRP(SYST) PARENT(MASTER)
       TEXT('SYSTEM TEST GROUP IN PROJECT PAYROLL')
```

The next step is to change the parent of the group TEST from the group MASTER to the group SYSTEST.

Example: The following CL command changes the parent group of the group TEST so that the project hierarchy structure looks like the After project hierarchy in Figure 13 on page 27.

Using the CHGGRP command

```
CHGGRP PRJ(PAYROLL) GRP(TEST) PARENT(SYSTEST)
```

Using the Programming Development Manager utility

Select option 2 (Change) on the Work with Groups Using PDM display.

If you want to change the project hierarchy by creating a new root group, simply create the group as you would any other group in this project, and specify PARENT(*NONE) on the CRTGRP command. The new root group becomes the parent of the previous root group.

When a group is added to the project hierarchy, the Application Development Manager/400 feature ensures that the promote path for all parts in groups lower down on the promote path is not disrupted. If you attempt to add a group to the project hierarchy, you would cause a part to be promoted along a different promote path from the one indicated on the Check Out Part (CHKOUTPART) command, and the CHGGRP command would fail.

To avoid possible disruption of the promote path, you should ask all developers to stop checking parts in and out before you insert a group in the project hierarchy. It is also a good idea to have all parts promoted to their target groups.

Note that a group cannot be its own parent. Neither can a group lower down in the hierarchy be identified as the parent or a group higher up in the hierarchy.

Changing the Promote Path in a Project Hierarchy

The promote code must represent a valid promote path in the existing project hierarchy. When the promote path is changed for a project hierarchy, this feature checks to see that a part is not prevented from being promoted to its target group.

Example: The following CL command changes the promote code for the group DEVELOPER1 in the project PAYROLL, causing the project to appear as in Figure 14.

```
CHGGRP PRJ(PAYROLL) GRP(DEVELOPER1) PRMCODE(*NONE)
```

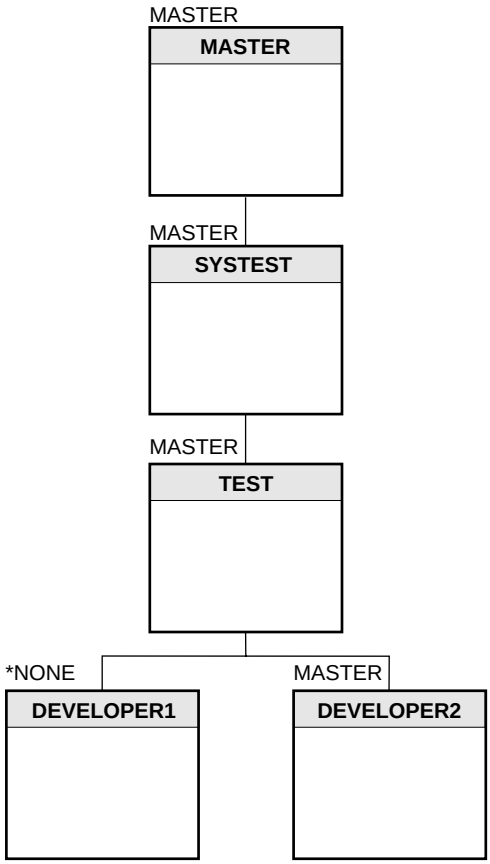

Figure 14. The Project PAYROLL with a Changed Promote Code

In this example, the promote code associated with the group DEVELOPER1 is changed so that the person working in that group can no longer promote parts. If a part in this group has a promote code of MASTER when the CHGGRP command is entered, the command fails because the part can no longer be promoted back to the target group.

If the promote code of a group in an existing project hierarchy is changed, this feature also ensures that parts in the group do not acquire a different target group on their promote path. For example, if the project hierarchy illustrated in Figure 15 is changed using the CHGGRP command to assign the V1 promote code to the group MASTER, the promote path for the part BIWKLY is extended because the part would have a new target group. The CHGGRP command would fail in this situation.

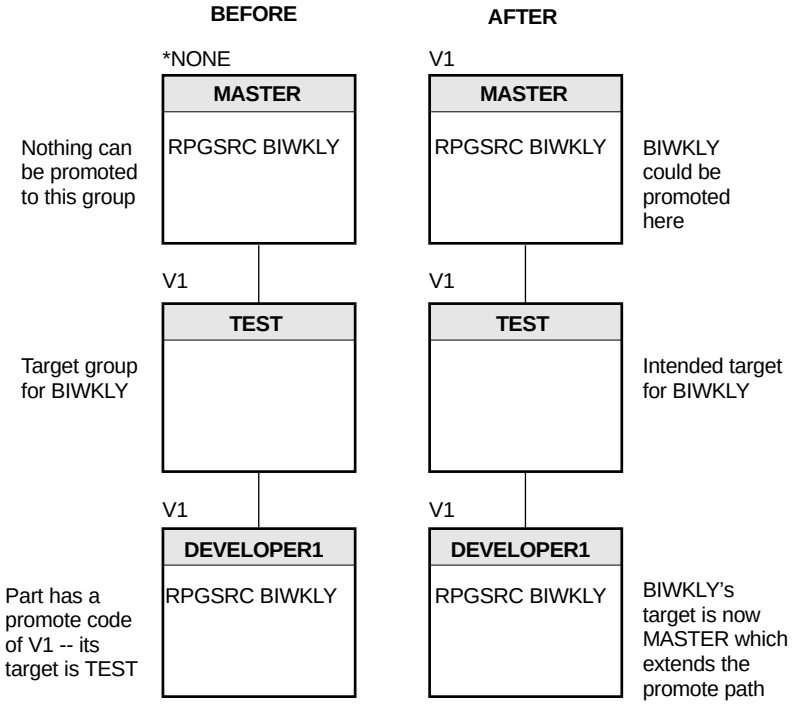

Figure 15. An Example Showing Why Promote Paths Cannot Be Extended

Deleting a Group (DLTGRP)

Use the Delete Group (DLTGRP) command to delete a group from the project hierarchy and delete the authorization information associated with the group. Only project administrators can do this.

You can use this command to remove a particular developer's group after the person leaves the project, or to remove certain groups associated with an old version of the project. You may no longer need development groups in the first version of your project if, for example, you are working on the third version.

Note

When a group is deleted, the archive library is also deleted. A library created outside of the Application Development Manager/400 feature is also deleted if its name is a valid archive library name. The archive and the group libraries are deleted only if their owner is QPRJOWN, otherwise, a warning is issued for the group being deleted. See “Archiving a Part” on page 94 for a description of valid archive library names.

Example: This example illustrates how to delete the group DEVELOPER1 from the project PAYROLL.

Using the DLTGRP command

```
DLTGRP PRJ(PAYROLL) GRP(DEVELOPER1)
```

Using the Programming Development Manager utility

Select option 4 (Delete) on the Work with Groups Using PDM display.

The Programming Development Manager utility provides you with a confirmation display when you use option 4. You are not given a warning if you use this command directly from the command line.

The DLTGRP command checks for the following conditions when deleting a group:

1. A group being deleted at the bottom of the project hierarchy must not have any parts in it with a promote code other than *NONE. Such a group can be deleted if all the parts it contains have a promote code of *NONE, and none of the parts is checked out.
2. Elsewhere in the project hierarchy, the following statements must be true.
   - There are no parts at all in the group.
   - There are no parts in the project hierarchy that use this group to reach its target group.
   - If the group is a root group, it must be empty. It must either be the only group in the hierarchy, or have only one group below it that becomes the new root group.

If you attempt to delete a part from a development group, or delete a group containing a part that has not been promoted to its target, the command fails.

The Programming Development Manager utility provides you with a confirmation display if you enter the DLTGRP command using option 4. You do not receive a warning if you use this command directly from the command line.

Enrolling Users in a Project Hierarchy

Once you create your project hierarchy, you need to give developers or administrators access to it. Project administrators enroll developers and other project administrators in projects and groups.

You should take the time to read the section in the *ADTS/400: Application Development Manager Introduction and Planning Guide* that lists some of the things to consider before you begin to enroll users.

Enrolling Developers and Project Administrators (ADDPRJUSR)

Use the Add Project User (ADDPRJUSR) command to enroll other AS/400 users in the project. Only project administrators can do this.

The AS/400 user profile is used to identify the person you want to enroll. You specify the user profile name on the USRPRF parameter of the ADDPRJUSR command. The user profile QSECOFR is an administrator for all projects.

The type of user you enroll can be determined by noting the value provided on the USRTYPE parameter. A user type of *DEVELOPER, for example, indicates that the person you are enrolling is an application developer and has read access to all the groups in the project hierarchy.

You grant update access to specific development groups by specifying *UPDATE on the ACCESS parameter and entering group names on the DEVELOPGRP parameter. Update access means that developers can promote parts from their development groups. These parameters are used together to indicate that the developer being enrolled requires update access to the groups specified. When ACCESS(*READ) is specified, the DEVELOPGRP parameter is ignored.

Example: This example illustrates how to give a developer read access to all parts in the project hierarchy and update access to all the parts in one development group.

Using the ADDPRJUSR command

```
ADDPRJUSR PRJ(PAYROLL) USRPF(SMITH) USRTYPE(*DEVELOPER) ACCESS(*UPDATE)
          DEVELOPGRP(DEVELOPER1)
```

Using the Programming Development Manager utility

Select option 41 (Add user) on the Work with Projects Using PDM display.

If no groups are identified on the DEVELOPGRP parameter, the developer cannot check parts in or out or promote parts in the project hierarchy.

Example: The following command gives a developer only read access to all parts in the project hierarchy.

```
ADDPRJUSR PRJ(PAYROLL) USRPRF(SMITH) USRTYPE(*DEVELOPER) ACCESS(*READ)
```

A user type of *ADMIN indicates that the person you are enrolling is a project administrator who has update access to all the groups in the project hierarchy, and who can use all Application Development Manager/400 commands. The user profile QSECOFR is an administrator for all projects.

Example: The following command illustrates how to enroll a project administrator.

```
ADDPRJUSR PRJ(PAYROLL) USRPRF(TREMBLAY) USRTYPE(*ADMIN)
```

The ACCESS and DEVELOPGRP parameters are ignored if you specify USRTYPE(*ADMIN), because update access to all groups in the project hierarchy is implied.

See “Changing User Enrollment Information (CHGPRJUSR)” on page 34 for information on how to change enrollment information for a user profile after a developer or administrator has been enrolled in a project.

Working with Project User Commands

You can work with the user profiles of project users by typing CL commands directly on any command line. You can also use the commands in a CL program, a REXX procedure, or through QCMDEXC. The project user commands are ADDPRJUSR, PRTPRJUSR, CHGPRJUSR, and RMVPRJUSR. You must be a project administrator to use the ADDPRJUSR, GHGPRJUSR, and RMVPRJUSR commands.

All the project user commands work with AS/400 user profiles. You must provide the user profile for each command.

You can also work with project users using the Programming Development Manager utility using the Work with Projects Using PDM display.

Printing User Enrollment Information (PRTPRJUSR)

Use the Print Project User (PRTPRJUSR) command to see a particular user profile's enrollment information for a project. Both project administrators and developers can do this.

You must specify which project you want to display user enrollment information about on the PRJ parameter. You can enter a specific AS/400 user profile name, or use *ALL on the USRPRF parameter to indicate that you want to see all the enrollment information for all people enrolled in a project.

The access level granted on the last ADDPRJUSR or CHGPRJUSR command is used in combination with the user-type value to create the report. If the user profile you specify on this command is for a developer who has update access to some groups, the groups to which he or she has update access are listed. If a developer has only read access to a project, or if the user profile you specify is for an administrator, ALL is printed in place of the list of groups on the report.

Example: The default value for the OUTPUT parameter, *PRINT, spools the report for printing. The report shown in Figure 16 on page 34 is spooled if you enter the following command.

Using the PRTPRJUSR command

```
PRTPRJUSR PRJ(PAYROLL) USRPRF(*ALL) OUTPUT(*PRINT)
```

Using the Programming Development Manager utility

Select option 46 (Print user) on the Work with Projects Using PDM display.

```
5716PW1   V3R7M0          Application Development Manager/400 - Print Project User   11/08/96    17:30:15         Page . . :  0001

------------------------------------------------------------------------------

Project . . . . . . . . . . . . . . . :   PAYROLL
User  . . . . . . . . . . . . . . . . :   TREMBLAY
User type . . . . . . . . . . . . . . :   *ADMIN
Project access level. . . . . . . . . :   *ALL


Authorized groups
*** ALL ***
------------------------------------------------------------------------------

Project . . . . . . . . . . . . . . . :   PAYROLL
User  . . . . . . . . . . . . . . . . :   SMITH
User type . . . . . . . . . . . . . . :   *ADMIN
Project access level. . . . . . . . . :   *ALL


Authorized groups
*** ALL ***
------------------------------------------------------------------------------

Project . . . . . . . . . . . . . . . :   PAYROLL
User  . . . . . . . . . . . . . . . . :   ROBERTS
User type . . . . . . . . . . . . . . :   *DEVELOPER
Project access level. . . . . . . . . :   *UPDATE


Authorized groups
DEVELOPER1
------------------------------------------------------------------------------

Project . . . . . . . . . . . . . . . :   PAYROLL
User  . . . . . . . . . . . . . . . . :   LALONDE
User type . . . . . . . . . . . . . . :   *DEVELOPER
Project access level. . . . . . . . . :   *READ


Authorized groups
*** ALL ***
           * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 16. A Sample of a Spooled File Produced by the PRTPRJUSR Command

Example: Alternatively, you can direct the output to an output file by specifying the following command. The library list is used to determine where to store the output file called FILE1, and the first member in the file receives the output.

```
PRTPRJUSR PRJ(PAYROLL) USRPRF(*ALL) OUTPUT(*OUTFILE)
          OUTFILE(*LIBL/FILE1) OUTMBR(*FIRST)
```

The record format of the output file is the same as that used in the system-supplied database file QALYPUSR in library QADM.

If the user profile you specify on this command is for a developer, one output file record is created for each group to which the developer has update access. If the user profile you specify is for an administrator, only one output file record is created.

Changing User Enrollment Information (CHGPRJUSR)

Use the Change Project User (CHGPRJUSR) command to change a particular user profile's enrollment information for a project. With this command you can change the user type (USRTYPE), the access authority (ACCESS), or the groups to which a developer has update access (DEVELOPGRP). You must ensure that the users whose enrollment information you change do not have any parts checked out to their development groups. Only project administrators can do this.

You can change the user profile of someone currently enrolled as a developer if that person becomes a project administrator or vice versa. You can remove a developer's update access to all development groups by changing his or her access to *READ. To remove a developer or administrator's authority from the project entirely, use the RMVPRJUSR command. The group names that you enter on the DEVELOPGRP parameter act as a replacement list for the groups to which the developer was previously authorized.

Examples: Assume that the following command was previously entered.

```
ADDPRJUSR PRJ(PAYROLL) USRPRF(ROBERTS) USRTYPE(*DEVELOPER)
          ACCESS(*UPDATE) DEVELOPGRP(DEVELOPER1)
```

The next example illustrates how to change update access for the developer ROBERTS from the group DEVELOPER1 to the group DEVELOPER2.

Using the CHGPRJUSR command

```
CHGPRJUSR PRJ(PAYROLL) USRPRF(ROBERTS) USRTYPE(*SAME)
          ACCESS(*SAME) DEVELOPGRP(DEVELOPER2)
```

Using the Programming Development Manager utility

Select option 42 (Change user) on the Work with Projects Using PDM display.

To allow a developer to update parts in another development group, you must specify all groups on the CHGPRJUSR command. For example, to give ROBERTS update access to both the DEVELOPER1 and DEVELOPER2 groups, you would enter the following command.

```
CHGPRJUSR PRJ(PAYROLL) USRPRF(ROBERTS) USRTYPE(*SAME)
          ACCESS(*SAME) DEVELOPGRP(DEVELOPER1 DEVELOPER2)
```

If the existing list of groups is a long one, press F4=Prompt first to prompt the command and retrieve the current list of groups before changing it.

Removing User Enrollment Information (RMVPRJUSR)

Use the Remove Project User (RMVPRJUSR) command to remove a user profile's enrollment for the project and the groups within the project. The user profile QSECOFR is an administrator for all projects, and can remove all project administrators from a project. Only project administrators can do this.

You use this command when an administrator or a developer leaves the project, or when you want to remove a user entirely from a project hierarchy.

Example: The following command removes ROBERTS from the project PAYROLL.

Using the RMVPRJUSR command

```
RMVPRJUSR PRJ(PAYROLL) USRPRF(ROBERTS)
```

Using the Programming Development Manager utility

Select option 44 (Remove user) on the Work with Projects Using PDM display.

The RMVPRJUSR command checks the parts back in and issues informational messages if there are any parts checked out to the user profile specified on the command. The administrator can use these messages to resolve any outstanding work items that this user, whether developer or administrator, was working on before using the RMVPRJUSR command. See “Recovering When a User Leaves the Project” on page 211 for information on how to clean up a development group when someone leaves the project.

Creating a Sample Project Hierarchy

This section describes a series of commands you can use to:

- Create a project similar to project TRAVEL described here
- Create the groups in the project hierarchy
- Display information about the project
- Enroll users in it
- Change some of their enrollment information

You can also create this project hierarchy using the displays the Programming Development Manager utility provides. At the end of this exercise, you will have created the project hierarchy illustrated in Figure 17 on page 37.

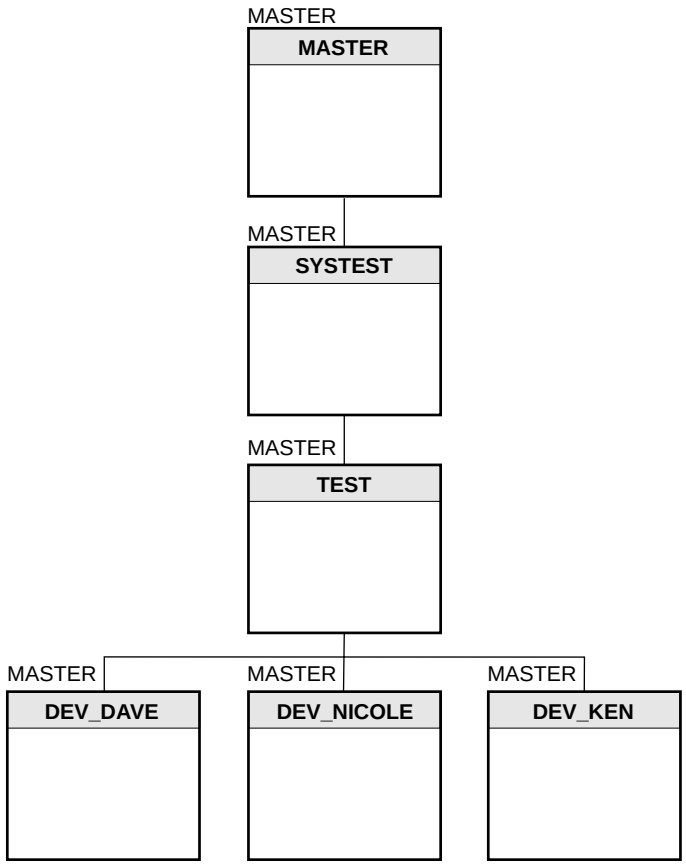

Figure 17. The Project Hierarchy for a Travel Application

Use the following steps to create this new project hierarchy:

1. Create the project using the CRTPRJ command. The following command creates the project TRAVEL. If you prefer, create the project with a different name so that you can keep it on your system for future practice.

   ```
   CRTPRJ PRJ(TRAVEL) SHORTPRJ(TRL) TEXT('TRAVEL DESTINATIONS APPLICATION')
   ```

   For more information on creating a project, see “Creating a Project (CRTPRJ)” on page 9.

2. Create the root group, called MASTER, in the project TRAVEL by using the following command.

   ```
   CRTGRP PRJ(TRAVEL) GRP(MASTER) SHORTGRP(MST) PARENT(*NONE)
          TEXT('MASTER GROUP IN PROJECT TRAVEL')
   ```

   Eventually, you want this group to contain all the production-level parts of your application. For more information on creating a group, see “Creating a Group (CRTGRP)” on page 16.

3. Create the system test group, called SYSTEST. Create this group with an appropriate short group name and a promote code of MASTER. Describe this group on the TEXT parameter. Remember to specify the parent for this group as MASTER.

   Repeat this step to create the group TEST. This group also has a promote code of MASTER, but its parent is the group SYSTEST.

4. Now you can add the development groups you need, again using the CRTGRP command. In this example, we need three development groups, one each for Dave, Nicole, and Ken. All three groups have the TEST group as their parent group and a promote code of MASTER. Call Dave's group DEV_DAVE, Nicole's group DEV_NICOLE, and Ken's group DEV_KEN.
5. Verify that the project hierarchy is what you want by using the PRTPRJ command. For more information on printing project information, see “Printing Information About a Project (PRTPRJ)” on page 11. Your report should look similar to the one shown in Figure 3 on page 11. If you are satisfied with your results, go to the next step. If you want to change the project hierarchy's information, try using the CHGGRP command to change the text description about one of the groups.
6. Now you need to enroll the developers to the project TRAVEL so they can begin their work. For the moment you want to enroll all three developers to have update access to their own groups only. Do this with the ADDPRJUSR command. Because you need to have real AS/400 user profiles to use this command, choose three of your co-workers to play the parts of Dave, Nicole, and Ken. For more information on enrolling project users, see “Enrolling Developers and Project Administrators (ADDPRJUSR)” on page 31.
7. After you have enrolled Dave, Nicole, and Ken to the project TRAVEL, you can verify their enrollment information by using the PRTPRJUSR command. For more information on printing this kind of information, see “Printing User Enrollment Information (PRTPRJUSR)” on page 33. Your report should look similar to the one shown in Figure 16 on page 34. Note that your user profile is also listed on your report because you have created the project and, therefore, you are also the project administrator.
8. You might decide that you need to designate a backup project administrator. Use the CHGPRJUSR command to do this. See “Changing User Enrollment Information (CHGPRJUSR)” on page 34 if you need more information on how to do this. Verify what you have done by using the PRTPRJUSR command again.

When you have finished, your project hierarchy should look like the one in Figure 17 on page 37. You may want to experiment further to gain more practice using these commands. When you have finished, remember to delete the project using the DLTPRJ command so that others may use the same example. To delete your project, enter the following command:

```
DLTPRJ PRJ(TRAVEL)
```

Chapter 4. Introducing Part Development

This chapter describes:

- How to list parts and how to display or print their contents.
- The commands you can use to obtain the names of the projects to which you are enrolled and the groups to which you are authorized within those projects. These commands help you know which parts you can work with.

To learn how part names are constructed and the rules associated with naming a part, see Appendix C, “Naming Rules.”

Listing Project and Group Names and Your Access

To determine in which projects you are enrolled and which groups you can access, use CL commands or the Programming Development Manager utility. Use the Query Project (QRYPRJ) command to obtain a list of the projects to which you are enrolled. You can also use WRKPRJPDM to see a listing of the projects to which you have access. Once you know the name of a project, use the Print Project User (PRTPRJUSR) command to print the information about your user profile's access to the project. The Print Project (PRTPRJ) command lets you print a report that shows all the groups in a project. You can also use WRKGRPPDM to see a listing of all the groups in the project.

These commands are described in Chapter 2, “Working with a Project” and Chapter 3, “Working with Groups in a Project Hierarchy.” Refer to Chapter 16, “Using the Programming Development Manager Utility” for more information on using the Programming Development Manager utility to list the projects in which you are enrolled.

Viewing a List of Parts (QRYPART)

If you know the name of the project to which you are enrolled, you can get a list of the parts in a particular group or search path in the project.

Using the QRYPART Command

```
QRYPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
```

Using the Programming Development Manager utility

Choose option 6 (Work with parts) on the PDM main menu. Specify the name of a project and group in the *Project* and *Group* prompts and press Enter. All the parts from the group specified up to the root group are listed. You can also use the WRKPARTPDM to get a listing of the groups to which you have access.

Required Parameters

Specify the project, group, type, and part to be queried.

For the TYPE and PART parameters, you can specify a generic type and name, all types and names, or a specific type and name.

For the GRP parameter, you can specify a specific name, or *ALL. When GRP(*ALL) is specified:

- The SCAN and the SCHPTH parameters on the QRYPART command are ignored and all the groups in the project are searched
- Duplicate parts appear in the Query Part report, since all of the versions of parts are listed

If you use generic type or part names (**generic**), all parts and types that match it are queried. Specify a subset of all the parts or types by qualifying the part name or type with an asterisk (*) or a question mark (?) where * denotes 0 or more characters and ? denotes exactly 1 character. For example, if you specify *C* on the TYPE parameter, all parts of a type that have the character C anywhere in the part type are processed, such as CBLINC or CINC. If you specify ?ABC on the PART parameter, all parts with 4-character names that end in ABC are processed, such as DABC. See Appendix B, “Part Types and Their Relationship to Commands” for a list of the part types allowed with the QRYPART command. User-defined part types are also allowed with the QRYPART command.

Because *ALL has a special meaning to the PART parameter, use **ALL to specify all parts ending in ALL.

Generic types and names are handled differently in the Programming Development Manager utility. See Chapter 16, “Using the Programming Development Manager Utility” for more information.

Optional Parameters

You can specify some optional information on the QRYPART command. You can indicate where the output of the command should go, whether the hierarchy should be scanned, and which search path should be used.

The ACCKEY parameter allows you to query all the parts checked out to a particular user. The default is ACCKEY(*ALL) and it queries all the parts regardless of who has checked them out.

Use the OUTPUT parameter to indicate where the output should go. The default is OUTPUT(*PRINT). The report is spooled to the print device for this job. OUTPUT(*OUTFILE) directs the output to an output file. An **output file** is a file that contains the result of some processing. The record format of the output file is the same as that used in the system-supplied database file QALYQPART in library QADM.

Example: You can direct the output to an output file by specifying a command similar to the following.

```
QRYPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
        OUTPUT(*OUTFILE) OUTFILE(*LIBL/FILE1) OUTMBR(*FIRST *REPLACE)
```

Example: As Figure 18 illustrates, the following command looks for the part RPGSRC BIWKLY in the project PAYROLL, first searching the group DEVELOPER1. The part is not found there, so the QRYPART command looks for it along the default search path (in this example, the default search path is DEVELOPER1, TEST, MASTER), and finds it in the group MASTER. The report in Figure 19 is spooled to a print device.

```
QRYPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
        OUTPUT(*PRINT) SCAN(*YES) SCHPTH(*DFT)
```

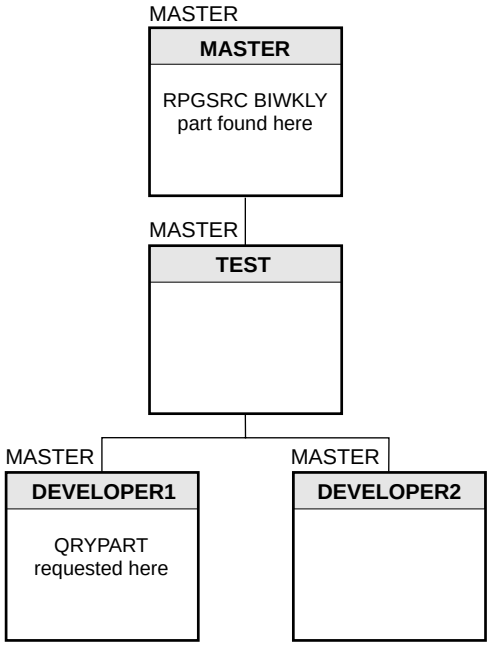

Figure 18. Querying a Part

```
5716PW1   V3R7M0              Application Development Manager/400 - Query Part      11/08/96    12:00:00         Page . . :  0001

Project . . . . . . . . . . . . . . . :   PAYROLL
Group . . . . . . . . . . . . . . . . :   DEVELOPER1
Type  . . . . . . . . . . . . . . . . :   RPGSRC
Part  . . . . . . . . . . . . . . . . :   BIWKLY
Access key (holder) . . . . . . . . . :   *ALL
Scan hierarchy. . . . . . . . . . . . :   *YES       1
Search path . . . . . . . . . . . . . :   *DFT       2

                                    Parts That Meet the Search Criteria

                                                                                  Date Last
Part        Type        Language    Group                                     changed     Holder      Text

BIWKLY      RPGSRC      RPG         MASTER                                    11/08/96    SMITH       Update payroll
                         3           4                                          5           6           7
                 * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 19. Sample of a Query Part Report

1 Specifies whether the project hierarchy is to be searched for the part specified.

2 The search path specified on the SCHPTH parameter.

3 The part's language.

4 The group in which the part was found.

5 The date when the contents of the part were last changed.

6 Shows the user profile (if any) of the person who has the part checked out. Blanks show the part is not checked out.

7 Shows any descriptive text for the part.

Displaying and Printing a Part (DSPPART)

Use the Display Part (DSPPART) command to display or print the contents or characteristics of a part. Table 1 lists the commands the DSPPART command calls and the various part types.

Example: Specify the name of the project, the group, the type, and the part. You can also choose where you want the output to be directed and whether to use the default search path to look for the part if it is not found in the specified group.

```
DSPPART PRJ(PAYROLL)
```

The TYPE parameter determines the command that is called to display or print a part.

Table 1 (Page 1 of 2). Commands Called to Display Parts of Given Types

Part type	Command called by DSPPART to display the part
BNDDIR	DSPBNDDIR
CLD	n/a
CMD	DSPCMD
DTAARA	DSPDTAARA
FILE with a language attribute of DSPF	STRSDA (STRSDA is called and the DSPFD command is called to print them.)
FILE with the language attribute of ICFF, LF, PF, or PRTF	DSPFD
JOBD	DSPJOBD
JOBQ	WRKJOBQ
MENU (UIM)	DSPMNUA
MODULE	DSPMOD
MSGF	DSPMSGD
MSGQ	WRKMSGQ
OUTQ	WRKOUTQ
PARTL[1]	DSPPFM
PGM	DSPPGM
PNLGRP	n/a
SCHIDX	n/a

Table 1 (Page 2 of 2). Commands Called to Display Parts of Given Types

Part type	Command called by DSPPART to display the part
Source, include, search path, build options parts, PRDDFN, PRDLOD, and SYSTEML parts	STRSEU
SRVPGM	DSPSRVPGM
VRPGBIN and VRPGTXT	DSPFD

Note:

1. For more information on displaying a part of type PARTL, see "Displaying a Part-List Part" on page 106.

The OUTPUT parameter determines where the output should go. Use the default OUTPUT(*) to have the output appear on your display. OUTPUT(*PRINT) spools it to the output queue of your print device.

The SCAN parameter determines whether to use the default search path to find the part if it is not found in the specified group. Use SCAN(*NO) if you want only the group specified on the GRP parameter to be searched. Use SCAN(*YES) if you want the groups in the default search path to be searched.

Chapter 5. Creating a Part

This chapter describes how to create a part. You can create a new part, or you can copy an existing part to a new part.

You can also create a part by importing an OS/400 object or source member into Application Development Manager/400 control. This is described in Chapter 6, “Importing an Application.”

Types of Parts

There are several categories of parts:

- Source that you can compile
- Output objects
- Include source members
- Objects such as data areas or message files that are not source and are not generated by compiling source
- Search path parts which contain a list of groups that specifies a search path
- Build options parts that contain lists of compile commands and their parameters
- Part-list parts that contain lists of other parts
- Product definition and product load parts used to package applications

Table 2 lists the type of parts that you can create. The column on the right shows the parts' associated languages. This table does not include the system-supplied part types that are created as the result of issuing the BLDPART, IMPPART, or CPYPART commands: PGM, FILE, CLD, CMD, PNLGRP, MENU, MODULE, and SRVPGM. Table 5 on page 61 lists AS/400 object types supported by the Application Development Manager/400 feature.

Parts of types that can be created by the BLDPART command cannot be created with the CRTPART command, except when parts that are created by the BLDPART command are parts that are stored in source members.

Table 2 (Page 1 of 3). Part Type and Language

Part type	Language
BLDOPT	*NONE
BNDDIR	*NONE
BNDSRC	BND
CBL36INC	CBL36
CBL36SRC	CBL36
CBLINC	CBL, SQLCBL
CBLLEINC	CBLLE, SQLCBLLE
CBLLESRC	CBLLE, SQLCBLLE
CBLSRC	CBL, SQLCBL

Table 2 (Page 2 of 3). Part Type and Language

Part type	Language
CINC	C, CLE, SQLC, SQLCLE
CLDSRC	CLD
CLLESRC	CLLE
CLPSRC	CLP
CLSRC	CL
CMDSRC	CMD
CSRC	C, CLE, SQLC, SQLCLE
DDSSRC	DSPF, ICFF, LF, PF, PRTF
DSPF36SRC	DSPF36
DTAARA	*NONE
FILE	SAVF
JOBD	*NONE
JOBQ	*NONE
MNU36SRC	MNU36
MSGF	*NONE
MSGF36SRC	MSGF36
MSGQ	*NONE
OCL36SRC	OCL36
OUTQ	*NONE
PARTL	*NONE
PNLINC	PNLGRP
PNLSRC	MENU, PNLGRP
PRDDFN	*NONE
PRDLOD	*NONE
REXXSRC	REXX
RPG36INC	RPG36
RPG36SRC	RPG36
RPGINC	RPG, SQLRPG
RPGLEINC	RPGLE, SQLRPGLE
RPGLESRC	RPGLE, SQLRPGLE
RPGSRC	RPG, SQLRPG
RPT36SRC	RPT36
SCHIDX	*NONE
SCHPTH	*NONE
SRT36SRC	SRT36
SYSTEML	*NONE
TXT36SRC	TXT36
TXTSRC	*NONE

Table 2 (Page 3 of 3). Part Type and Language

Part type	Language
VRPGBIN	VRPG
VRPGTXT	VRPG

Creating a New Part (CRTPART)

Use the Create Part (CRTPART) command to create a new part in a group in the project hierarchy. A part of a given name and type can only be created if no part with the same name and type already exists in the group, and if no part with the same name and type can be promoted into the default search path for that group. If a part of the same name and type already exists in the default search path, you must specify a promote code of *NONE when you create the part. Consider Figure 20:

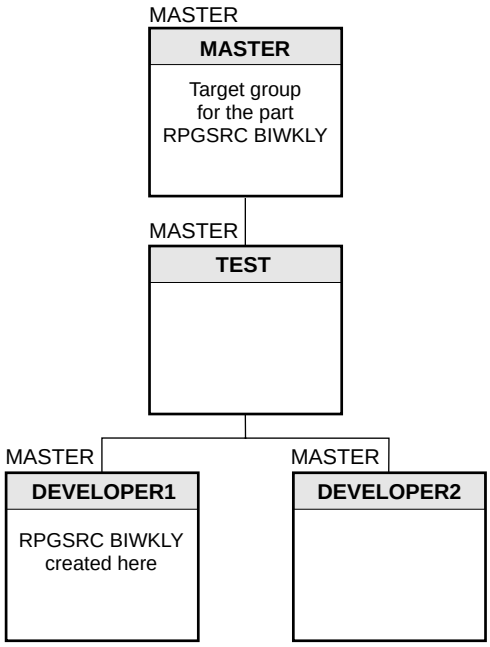

Figure 20. Creating a Part

If, for example, you have created a part called RPGSRC BIWKLY in the group DEVELOPER1 with a promote code of MASTER, so that the part's target group is MASTER, the only way you could create a part of the same name and type in the group DEVELOPER2, would be by specifying a promote code of *NONE. This avoids the problem of having two parts with the same name and type that can potentially both be promoted to the same group.

Required Parameters

Specify the project, group, type, and part name of the part being created. You must have update access to the group in which the new part is being created. See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types that are allowed with the CRTPART command.

You can create parts of type PARTL to track your changes, giving them a consistent naming convention. PART(*GENERATE) may only be specified if TYPE(PARTL) was specified. The first part-list name generated will be nnnn000001, where nnnn is the short project name. Subsequently, the generated part-list names will be nnnn000002, nnnn000003, and so on. If the generated name exists, the CRTPART command will continue to increment the number appended to the name up to a fixed number of times until it finds a part-list name that does not exist.

When you create a part, it is not checked out to your user profile.

Optional Parameters

The LANG parameter is where you specify a language for the part. Specify a language or use the default LANG(*DFT). *DFT is replaced with the first language listed alphabetically that has been defined for the type of the part being created. Specify LANG(*NONE) if no language is required. (See Table 2 on page 45 for a list of part types and associated languages.)

If the part you are creating is not a source part, such as a part of type MSGF, DTAARA, JOBD, or MSGQ, you can specify whether you want to be prompted for the appropriate create command. Table 5 on page 61 lists all the AS/400 object types supported by the Application Development Manager/400 feature.

If, for example, you are creating a part of type DTAARA, and you choose PRMPT(*NO), the Create Data Area (CRTDTAARA) command defaults are used. The data area is *CHAR with a length of 256 bytes. You cannot change the attributes of the data area after it is created. Input such as the object and library name is automatically provided.

If you choose PRMPT(*YES) in this case, the CRTDTAARA command is prompted. Information that is automatically filled in, such as object and library name, cannot be changed. The data area can be any type and size.

If the part you are creating is of type BLDOPT, the CRTPART command copies a system-supplied BLDOPT source member into the source member for this new part. See “Using Build Options” on page 157 for information about these parts.

You can create parts of type FILE and language SAVF for savefiles.

If the part you are creating is of type SCHPTH, the CRTPART command places the default search path from the specified group into the source member for this new part. See “Creating a Search-Path Part” on page 112 for information about these parts.

If the part you are creating is of type SCHIDX, the CRTPART command fills in the TITLE parameter with the name of the part you are creating. If this is not satisfactory to you, you should specify PRMPT(*YES) on the CRTPART command to be prompted for the Create Search Index (CRTSCHIDX) command.

The PRMCODE parameter is where you specify whether you want the part to be promoted. (Not all parts are; for example, you may decide to create a part just for test purposes and have no plans to keep it as a part of your application.) If you do intend to promote the part, choose the default PRMCODE(*GRP). This causes the promote code defined for the group specified on the GRP parameter to be assigned to the new part. If you do not want to promote the part, choose PRMCODE(*NONE), and the part cannot be promoted. If you need to look up the promote code for a group, use the Print Project (PRTPRJ) command.

This feature ensures that the part being created does not already exist and that it remains unique. Note that the EXTEND parameter on the PRMPART command allows output parts from the build process to be promoted along with the associated source parts. Output parts have a promote of *NONE.

The SRCFILE parameter is where you specify the source file where the part is to be stored as a source member, if applicable. Choose SRCFILE(*TYPE) to use the system-supplied default source file for that type of part, or enter the name of a specific source file. If you create an include part using the CRTPART or IMPPART command, with the SRCFILE parameter different from *TYPE, you must ensure that your include parts are qualified with a file name in the source. Table 3 shows part types and the corresponding AS/400 default source files in which they are stored. You can also specify the name of a user-defined type that is stored in an AS/400 source file member (SYSTYPE(*MBR) specified on the ADDADMTYPE) command.

Table 3 (Page 1 of 2). Part Types and Their Corresponding AS/400 Default Source Files

Type	**Default Source File**
BLDOPT	QBLDOPTSRC
BNDSRC	QSRVSRC
CBL36INC	QS36SRC
CBL36SRC	QS36SRC
CBLINC	QLBLSRC
CBLLEINC	QCBLLESRC
CBLLESRC	QCBLLESRC
CBLSRC	QLBLSRC
CINC (of language C, CLE, SQLC, SQLCLE)	H
CLDSRC	QCLDSRC
CLLESRC	QCLLESRC
CLPSRC	QCLSRC
CLSRC	QCLSRC
CMDSRC	QCMDSRC
CSRC (of language C, CLE, SQLC, SQLCLE)	QCSRC
DDSSRC	QDDSSRC
DSPF36SRC	QS36SRC
MNU36SRC	QS36SRC
MSGF36SRC	QS36SRC

Table 3 (Page 2 of 2). Part Types and Their Corresponding AS/400 Default Source Files

Type	Default Source File
PNLINC	QPNLSRC
PNLSRC (of language MENU)	QMNUSRC
PNLSRC (of language PNLGRP)	QPNLSRC
PRDDFN	QDFNSRC
PRDLOD	QLODSRC
OCL36SRC	QS36SRC
REXXSRC	QREXSRC
RPG36INC	QS36SRC
RPG36SRC	QS36SRC
RPGINC	QRPGSRC
RPGLEINC	QRPGLESRC
RPGLESRC	QRPGLESRC
RPGSRC	QRPGSRC
RPT36SRC	QS36SRC
SCHPTH	QSCHPTHSRC
SRT36SRC	QS36SRC
SYSTEML	QSYSTEMSRC
TXT36SRC	QS36SRC
TXTSRC	QTXTSRC

Specifying a PARTL name in the PARTL parameter causes the CRTPART command to automatically add the part being created to the part-list part that you specified. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being created will not be added to a part-list part.

A part-list part must be specified if:

- The group specified on the command was created with PARTLREQ(*YES)
- The type specified is not PARTL

If a part-list part is specified on the CRTPART command, then:

- The part-list part must exist in the default search path starting from the group in which the part is created.
- The created part name is added to the specified part-list part, if it does not already exist. (This may happen even if the command failed to create the part).

If the type specified is PARTL, then the PARTL parameter is ignored. In the other cases, the PARTL parameter results in the specified part-list part being updated.

Use the TEXT parameter to enter some descriptive text for the new part. If you do not enter any text, the default *BLANK is used, and the description is left blank.

Examples: These examples illustrate how to create a part called PAYROLL DEVELOPER1 RPGSRC BIWKLY.

Using the CRTPART command

```
CRTPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
        LANG(RPG) PRMPT(*NO) PRMCODE(*GRP) SRCFILE(QRPGSRC) TEXT(*BLANK)
```

Using the Programming Development Manager utility

Press F6=Create on the Work with Parts Using PDM display.

The part is promoted and stored as a source member in QRPGSRC, which is the default source file for a part of type RPGSRC. This is the same as specifying SRCFILE(*TYPE).

The following command creates the part MNTHLY in the group DEVELOPER2 in the project PAYROLL.

```
CRTPART PRJ(PAYROLL) GRP(DEVELOPER2) TYPE(CSRC) PART(MNTHLY)
        LANG(C) PRMPT(*NO) PRMCODE(*NONE) SRCFILE(QCSRC) TEXT(*BLANK)
```

The promote code *NONE indicates that the part cannot be promoted. The part is stored as a source member in QCSRC, which is the default source file for a part of type CSRC. This is the same as specifying SRCFILE(*TYPE).

Copying a Part (CPYPART)

Another way to create a new part is to copy an existing one to a specified group and project. Use the Copy Part (CPYPART) command to do this.

A part of a given name and type can only be copied to a group that does not already contain a part of the same name and type. If a part of the same name and type already exists in the default search path of the group to contain the copied part, you must specify a promote code of *NONE when you copy the part.

You must have read or update access to the group containing the part being copied and update access to the group receiving the copy.

After you have copied a part, it is not checked out to your user profile.

When you copy a logical file part, the physical file on which it is based must be copied first or must already exist in the receiving group. If you are copying a physical file, specify on the DATA parameter whether you want to copy the data in the physical files as well.

If you want to change attributes of the copied part, such as its language or text, use the Change Part Information (CHGPARTINF) command after the copy is complete.

The Part You Want to Copy

On the CPYPART command, you specify *from* criteria and *to* criteria. The *from* information consists of the project, group, type, and part. Use the FROMPRJ, FROMGRP, FROMTYPE, and FROMPART parameters. See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types that are allowed with the CPYPART command.

Use FROMTYPE(*ALL) if you want to copy parts of all types, and FROMPART(*ALL) if you want to copy all parts of the types specified on the FROMTYPE parameter.

If the part you want to copy is not found in the group you specified, use the SCAN parameter. The SCAN parameter specifies whether to search the default search path for the part being copied if it is not found in the specified group. The default is *YES; all groups in the path from the specified group up to the root group are searched.

The Part You Want to Create

The TOPRJ, TOGRP, TOTYPE, and TOPART parameters let you specify the project, group, type, and part of the part being created. If you do not fill in these parameters, the same names that were specified on the *from* parameters are used. If you specified *ALL on the FROMTYPE and FROMPART parameters, you must use the defaults TOTYPE(*FROMTYPE) and TOPART(*FROMPART).

If the original part is an AS/400 source member, the TOTYPE parameter must be a source type. For example, if the original part was of type RPGSRC (AS/400 source member type of RPG), it could be copied to a part of type RPGSRC, CSRC, or RPGINC.

When copying a source part to a different source type, such as RPGSRC to CSRC, or TXTSRC to CBLSRC, an appropriate language is assigned. For example, if you specify CBLSRC on the TOTYPE parameter, the language CBL is assigned to the part; if you specify DDSSRC on the TOTYPE parameter, the language DSPF is assigned. For user-defined types, the default language assigned is the first language, listed alphabetically, that has been defined for the type of the part being created.

If the original part is not an AS/400 source member, the new part type must be stored in the same object type as the part type specified on the FROMTYPE parameter. (See Table 4 on page 59 for a list of AS/400 source member types and corresponding Application Development Manager/400 part types.) See Appendix B for a list of part types that are allowed with the CPYPART command. User-defined types are also allowed with the CPYPART command.

Parts of type VRPGTXT or VRPGBIN can be copied from one group to another or from one project to another using this command as long as the part name remains the same.

Use the PRMCODE code parameter to specify whether the new part is to be promoted. This parameter applies to several commands and is described in the section called “Promote Codes” on page 88.

Use the SRCFILE parameter to specify the source file where the part is to be stored as a source member. This parameter applies only to the part types valid for the CPYPART listed in Appendix B, “Part Types and Their Relationship to Commands” on page 247. The default is *SAME. The source file name is the same as that used by the original part. Use *name* to identify a specific source file, or the default, *TYPE, to specify that the system-supplied default source file is to be used. (See Table 3 on page 49 for a list of default source files.)

Use the DATA parameter to specify whether to copy data when copying physical files. The default is *NO.

Specifying a PARTL name in the PARTL parameter causes the CPYPART command to automatically add the part being created to the specified part-list part. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being created will not be added to a part-list part.

A part-list part must be specified if:

- The group specified on the command was created with PARTLREQ(*YES)
- The type specified is not PARTL

If a part-list part is specified on the CPYPART command, then:

- The part-list part must exist in the default search path starting from the copy target group.
- The created part name is added to the specified part-list part, if it does not already exist. (This may happen even if the command failed to create the part).

If the type specified is PARTL, then the PARTL parameter is ignored. In the other cases, the PARTL parameter results in the specified part-list part being updated.

In the project hierarchy illustrated in Figure 21 on page 54, the part BIWKLY is copied from the group TEST to the group DEVELOPER3.

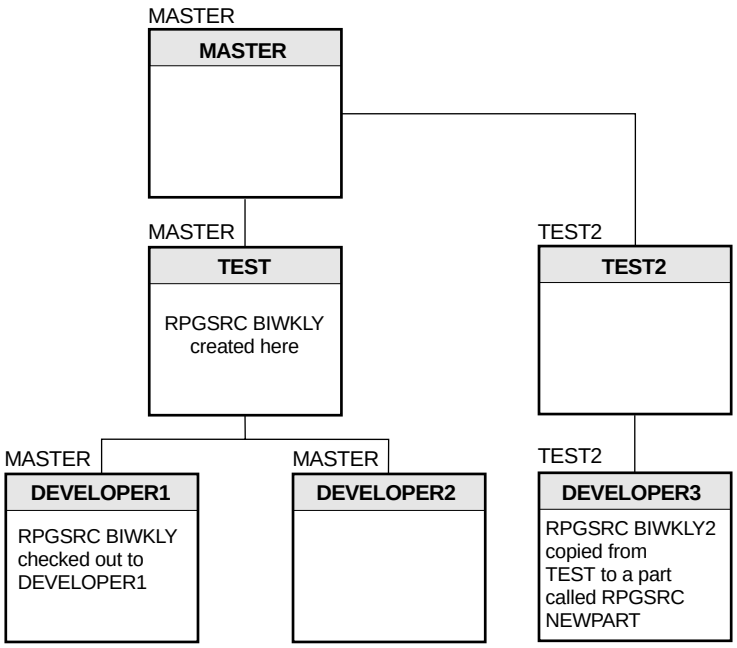

Figure 21. Copying a Part When it Exists in Several Groups

Example: The following command copies the part BIWKLY.

Using the CPYPART command

```
CPYPART FROMPRJ(PAYROLL) FROMGRP(TEST) FROMTYPE(RPGSRC)
        FROMPART(BIWKLY) TOPRJ(*FROMPRJ) TOGRP(DEVELOPER3)
        TOTYPE(*FROMTYPE) TOPART(NEWPART) SCAN(*YES)
        PRMCODE(*GRP) SRCFILE(*TYPE)
```

Using the Programming Development Manager utility

Select option 3 (Copy) on the Work with Parts Using PDM display.

As specified on the command, the part RPGSRC BIWKLY is copied from the group TEST to the part RPGSRC NEWPART in the group DEVELOPER3. The copy is successful because no part of this type and name exists in either the DEVELOPER3 group or in the promote path for the DEVELOPER3 group. If this part did exist in the promote path, it could only be checked out, not copied. For example, this would be the case if a part called RPGSRC NEWPART existed in the group MASTER.

Example: The following command copies RPGSRC BIWKLY from the group DEVELOPER1 to the group DEVELOPER3.

```
CPYPART FROMPRJ(PAYROLL) FROMGRP(DEVELOPER1) FROMTYPE(RPGSRC)
        FROMPART(BIWKLY) TOPRJ(*FROMPRJ) TOGRP(DEVELOPER3)
        TOTYPE(*FROMTYPE) TOPART(*FROMPART) SCAN(*NO)
        PRMCODE(*GRP) SRCFILE(*TYPE)
```

The type and name of the new part are the same as those of the copied part. The default on the PRMCODE parameter, *GRP, is specified. SRCFILE(*TYPE) specifies that the part is stored in the default source file QRPGSRC. This example assumes that you have update access to the DEVELOPER3 group.

If the part is not found in the group DEVELOPER1, the command fails because SCAN(*NO) was specified. Use the CHKOUTPART command if the part exists in the default search path from the group DEVELOPER3 (DEVELOPER3, TEST2, or MASTER), regardless of the promote code specified. Refer “Checking a Part Out (CHKOUTPART)” on page 71 for more information on the CHKOUTPART command.

Chapter 6. Importing an Application

Existing applications, pieces of an application, and individual members or objects can be brought under the control of the Application Development Manager/400 feature. This feature allows a developer or administrator to **import** objects or source members into a project hierarchy. To import means to copy one or more components of an application from an AS/400 library into an Application Development Manager/400 project hierarchy. Use the Import Part (IMPPART) command to do this.

This chapter describes how to import individual objects or an entire application into a project hierarchy and provides some step-by-step information on how to do this. Before you import an application, you should take the time to read the section on importing an application in the *ADTS/400: Application Development Manager Introduction and Planning Guide* that lists some of the things you should consider.

Importing into a Project Hierarchy

Typically, the administrator uses the IMPPART command to copy an entire application or a large piece of an application, while a developer is more likely to copy just a few source members or objects. Administrators can import objects or source members into any group in the project hierarchy; developers can import them into groups to which they have update access.

You must have sufficient AS/400 authorization to the objects and files you want to import. You need *USE authorization to the source file to import its members and *USE and Object Management authorization to any object you want to import. The IMPPART command fails if you do not have the correct authorization to the objects you attempt to import.

If your application is spread across several libraries, you can import all the objects and members in one library first, and then continue to the next library until all the components of your application have been successfully imported.

If REPLACE(*NO) is specified, then the IMPPART command behaves as before. If a part of the same name and type already exists in the default search path of the group that will contain the imported part, you must specify a promote code of *NONE when you import the part. The part's text description is updated to reflect the text description of the object or source member being imported. When you import a part, it is not checked out to your user profile.

If REPLACE(*YES) is specified, and if the part of the same name and type exists and is already checked out in the target group by the same user, then the part is replaced with the imported part. If the part was not checked out, and it exists in the default hierarchy, then the part is checked out and then replaced. The part thus checked out is checked back in when the import is complete.

If ARCHIVE(*YES) is also specified with REPLACE(*YES), then the original version of the part is archived.

To maintain Application Development Manager/400 integrity, all authorization to an imported object is changed once it becomes a part in a project. Public authorization to an imported object is set to *EXCLUDE. Private authorization to an imported object is set using the AS/400 authorization lists. See *Security – Reference*, for more information.

If you use the Programming Development Manager utility, you can use option 38 (Import) on the Work with Groups display to import members and objects. You can also use the user-defined options called IM and IO to help you import. IM imports members and IO imports objects. See "Working with User-Defined Options" on page 237 for information on how to do this.

Usually, parts that are created by the BLDPART command should not be imported. The BLDPART command needs to be able to compile imported parts to learn about the relationships among the parts that have been imported and as a result, the output parts are recreated.

If you import an output part, a warning is issued in the build report when the BLDPART tries to build it, even though a source part with the same name may exist in the search path of the build. In this instance, you will need to delete the output part, and issue the BLDPART command again.

If you want to import a logical file, you must first import the physical file on which it is based into the same group. It is not necessary to do this when importing *ALL objects as the IMPPART command ensures that physical files are copied before logical files.

Journaled files that are imported will *not* be journaled in the project hierarchy.

The File or Object You Want to Import

On the IMPPART command you must specify *from* and *to* criteria. The *from* information consists of the AS/400 library and object that represent the information you want to import. If you are importing a source file, you also indicate the member name. Use the OBJ, OBJTYPE, and MBR parameters to identify which AS/400 objects you want to import.

Specify the name of the library on the OBJ parameter. The special values *LIBL and *CURLIB, or a specific library name are supported.

You also specify the name of the object on the OBJ parameter or use a generic name. The generic object can be specified with the format ABC*, where all objects or files beginning with the characters you specify, such as ABC, are processed. A specific object name imports one object. The special value *ALL imports all objects found in the specified library. Note, however, that you cannot import all objects found in the library list by specifying OBJ(*LIBL/*ALL) on the IMPPART command. The special value *ALL can only be used with one library at a time. Any objects that have a user-defined type and an appropriate *CPY action defined for them are imported as well.

The OBJTYPE parameter indicates which type of AS/400 object you want to import. Only AS/400 object types that are supported by the Application Development Manager/400 feature can be imported. Table 4 on page 59 lists the supported source member types and Table 5 on page 61 lists the supported object types. You can also import user-defined part types.

Table 4 (Page 1 of 2). AS/400 Source Member Types Supported by the Application Development Manager/400 Feature

AS/400 Object Type	AS/400 Member Type	Part Type	Part Language
*FILE		VRPGTXT[1]	VRPG
*FILE	BND	BNDSRC	BND
*FILE	C[2]	CINC	C
*FILE	C[2]	CINC	CLE
*FILE	C[3]	CSRC	C
*FILE	C[3]	CSRC	CLE
*FILE	CBL[4]	CBLINC	CBL
*FILE	CBL[5]	CBLSRC	CBL
*FILE	CBL36	CBL36INC	CBL36
*FILE	CBL36	CBL36SRC	CBL36
*FILE	CBLLE[4]	CBLLEINC	CBLLE
*FILE	CBLLE	CBLLESRC	CBLLE
*FILE	CL	CLSRC	CL
*FILE	CLD	CLDSRC	CLD
*FILE	CLLE	CLLESRC	CLLE
*FILE	CLP	BLDOPT[6]	*NONE
*FILE	CLP	CLPSRC	CLP
*FILE	CLP	PRDDFN[6]	*NONE
*FILE	CLP	PRDLOD[6]	*NONE
*FILE	CMD	CMDSRC	CMD
*FILE	DSPF	DDSSRC	DSPF
*FILE	DSPF36	DSPF36SRC	DSPF36
*FILE	ICFF	DDSSRC	ICFF
*FILE	LF	DDSSRC	LF
*FILE	MENU	PNLSRC	MENU
*FILE	MNU36	MNU36SRC	MNU36
*FILE	MSGF36	MSGF36SRC	MSGF36
*FILE	OCL36	OCL36SRC	OCL36
*FILE	PF	DDSSRC	PF
*FILE	PNLGRP[4]	PNLINC	PNLGRP
*FILE	PNLGRP	PNLSRC	PNLGRP
*FILE	PRTF	DDSSRC	PRTF
*FILE	REXX	REXXSRC	REXX
*FILE	RPG[4]	RPGINC	RPG
*FILE	RPG	RPGSRC	RPG
*FILE	RPG36	RPG36INC	RPG36
*FILE	RPG36	RPG36SRC	RPG36

Table 4 (Page 2 of 2). AS/400 Source Member Types Supported by the Application Development Manager/400 Feature

AS/400 Object Type	AS/400 Member Type	Part Type	Part Language
*FILE	RPGLE[4]	RPGLEINC	RPGLE
*FILE	RPGLE	RPGLESRC	RPGLE
*FILE	RPT36	RPT36SRC	RPT36
*FILE	SQLC[2]	CINC	SQLC
*FILE	SQLC[2]	CINC	SQLCLE
*FILE	SQLC[3]	CSRC	SQLC
*FILE	SQLC[3]	CSRC	SQLCLE
*FILE	SQLCBL[4]	CBLINC	SQLCBL
*FILE	SQLCBL[5]	CBLSRC	SQLCBL
*FILE	SQLRPG[4]	RPGINC	SQLRPG
*FILE	SQLRPG	RPGSRC	SQLRPG
*FILE	SRT36	SRT36SRC	SRT36
*FILE	TXT	SCHPTH[6]	*NONE
*FILE	TXT	SYSTEML	*NONE
*FILE	TXT36	TXT36SRC	TXT36
*FILE		TXTSRC[7]	*NONE

Notes:

1. The member type of each member in a VRPGTXT part is set by VRPG based on its content, and will be retained by the IMPPART command.
2. If you import a source file called H, the parts will be created with type CINC. If the member type is C, then the language of the imported part will be CLE. Similarly, if the member type is SQLC, then the part language will be SQLCLE. If you want to import CINC parts with language C or SQLC, then you must specify the language explicitly.
3. If the member type is C and the part type is CSRC, then the language of the imported part will be CLE. If you want to import this part with language C or SQLC, then you must specify the language explicitly. Similarly, member type SQLC will have language SQLCLE.
4. Includes can be imported, but there is no information on the AS/400 system to distinguish source from includes. When importing RPG, SQLRPG, CBL, SQLCBL, or PNLSRC members, Application Development Manager/400 parts take the default part type that corresponds to the source member. To import these members into a part of type RPGINC, RPGLEINC, CBLINC, CBLLEINC, or PNLINC you must specify the part type explicitly or use the CVTPART command to change the type and language for the part.
5. If you import a member of type CBL or SQLCBL, and it contains more than one program, the BLDPART command compiles only the first program. The other programs are ignored, and a warning message is issued in the build report.
6. Special source files created by Application Development Manager/400 are QBLDOPTSRC, QSCHPTHSRC, QDFNSRC, QLODSRC, and QSYSTEMSRC. If you import members from these source files, parts of appropriate types are created. For example, BLDOPT parts are created when the source members are imported from QBLDOPTSRC.
7. All members whose type is not listed in the table are imported as parts of type TXTSRC and language *NONE. The member type does not change when the member is imported.

Table 5 (Page 1 of 2). AS/400 Object Types Supported by the Application Development Manager/400 Feature

AS/400 Object Type	AS/400 Object Attribute	Part Type	Part Language
*BNDDIR		BNDDIR	*NONE
*CLD		CLD	CLD
*CMD		CMD	CMD
*DTAARA		DTAARA	*NONE
*FILE	DSPF	FILE	DSPF
*FILE	ICFF	FILE	ICFF
*FILE	LF	FILE	LF
*FILE	PF	FILE	PF
*FILE	PF	PARTL	*NONE
*FILE	PF	VRPGBIN	VRPG
*FILE	PRTF	FILE	PRTF
*FILE	SAVF	FILE	SAVF
*JOBD		JOBD	*NONE
*JOBQ		JOBQ	*NONE
*MENU	UIM	MENU	*NONE
*MODULE	CBLLE	MODULE	CBLLE
*MODULE	CLE	MODULE	CLE
*MODULE	CLLE	MODULE	CLLE
*MODULE	RPGLE	MODULE	RPGLE
*MSGF		MSGF	*NONE
*MSGQ		MSGQ	*NONE
*OUTQ		OUTQ	*NONE
*PNLGRP		PNLGRP	*NONE
*PGM	C	PGM	C
*PGM	CBL	PGM	CBL
*PGM	CBL36	PGM	CBL36
*PGM	CBLLE	PGM	CBLLE
*PGM	CLE	PGM	CLE
*PGM	CLLE	PGM	CLLE
*PGM	CLP	PGM	CLP
*PGM	RPG	PGM	RPG
*PGM	RPG36	PGM	RPG36
*PGM	RPGLE	PGM	RPGLE
*SCHIDX		SCHIDX	*NONE
*SRVPGM		SRVPGM	*NONE
*SRVPGM	CBLLE	SRVPGM	CBLLE
*SRVPGM	CLE	SRVPGM	CLE

Table 5 (Page 2 of 2). AS/400 Object Types Supported by the Application Development Manager/400 Feature

AS/400 Object Type	AS/400 Object Attribute	Part Type	Part Language
*SRVPGM	CLLE	SRVPGM	CLLE
*SRVPGM	RPGLE	SRVPGM	RPGLE

Use the special value *ALL on the OBJTYPE parameter to show that you want to copy all the supported objects in a particular library. This value cannot be used with a library list, but only with one specific library at a time.

The special value *SRC indicates that you want to copy only objects that are source physical files. When you specify *SRC, all objects that have the object attribute of PF and are source physical files are imported. Table 4 shows how the AS/400 member types correspond to the Application Development Manager/400 part types. Note that any members of member types not listed in the table are also imported, but as parts of type TXTSRC. This value cannot be used with a library list, but only with one specific library at a time.

The special value *NONSRC shows that you want to copy all objects that are *not* source physical files. When you specify *NONSRC, all objects that match the object types listed in Table 5 on page 61 are imported. This value cannot be used with a library list, but only with one specific library at a time.

You can indicate a specific object type on the OBJTYPE parameter. For example, you can specify OBJTYPE(*PGM) to import all the programs, or OBJTYPE(*MSGF) to import all the message files. If you specify OBJTYPE(*FILE), all the objects with a type of *FILE and an attribute of PF (both data and source physical files), LF, DSPF, PRTF, SAVF, or ICFF are imported. The manner in which source files are imported depends on what you specify on the TYPE parameter. If you specify TYPE(*FILE), even source files are imported as physical data files. To import source files as source, that is, to import each member separately, use the default TYPE(*OBJTYPE).

When you are importing a source physical file, you can specify a specific source member on the MBR parameter, or indicate that you want to import all the members within the source file by using the special value *ALL. You can also use a generic name. The generic member name can be specified with a format such as ABC*, where all members beginning with the characters ABC are processed.

The value MBR(*ALL) is required if you specify *FILE, *SRC, or *ALL on the OBJTYPE parameter when OBJ(*ALL) is also specified. When importing only one member from a source file, specify OBJ(*filename*), OBJTYPE(*SRC) (or OBJTYPE(*FILE)), and MBR(*member-name*).

The Part You Want to Create

You must specify some information that indicates where you want the AS/400 object or source member to be imported.

Required Parameters

A part is created in the project and group you indicate, based on an AS/400 object or source member. Specify the project name on the PRJ parameter and the group name on the GRP parameter. These two parameters are required.

Specify the Application Development Manager/400 part type on the TYPE parameter. The part type defaults to the Application Development Manager/400 equivalent of the AS/400 object type when you specify the special value *OBJTYPE and when the object you are importing is not a source physical file. For example, if you import a *PGM object and specify TYPE(*OBJTYPE), the Application Development Manager/400 type is PGM.

If the object you are importing is a source physical file, each member becomes a separate part. For these parts, the part type defaults to the Application Development Manager/400 equivalent of the AS/400 object attribute and member type when you specify the special value *OBJTYPE. Refer to Table 4 and Table 5 to see how AS/400 types and part types relate.

You can also specify a specific part type. However, you cannot provide Application Development Manager/400 part types on the TYPE parameter that do not make sense for the AS/400 object imported. For example, you cannot import an object of type *PGM into a part of type RPGSRC. The IMPPART command fails if you attempt to do this.

You must specify VRPGTXT or VRPGBIN on the TYPE parameter of the IMPPART command in order to import VRPG source or binary objects. If a source file is imported and a generic part type is specified, the members in the source file are imported individually. If a physical file is imported and a generic part type is specified, the physical file is imported as a part of type *FILE.

Example: The following command imports a source member with C/400 source statements and creates a part HELLO of type CSRC. The part language is set to CLE, if the source member attribute of HELLO was C.

Using the IMPPART Command

```
IMPPART OBJ(*CURLIB/CSRC) OBJTYPE(*SRC) MBR(HELLO) PRJ(PAYROLL)
        GRP(MASTER) TYPE(CSRC) PART(*NAME)
```

Using the Programming Development Manager utility

Select option 38 (Import) on the Work with Groups Using PDM display.

If you import a source member containing C source with attributes C or SQLC, the parts of type CSRC are created (unless CINC type is explicitly specified on the TYPE parameter), and with the part language of CLE or SQLCLE respectively.

Specify the name of the part on the PART parameter. Choosing the part name default, *NAME, causes the source member name or the object name to become the name of the new part. Specify a part name if you want to explicitly name the part you are importing, either to give it a new name or because you are only importing objects or source members with that name.

Example: The member BIWKLY is imported and renamed EMPMST.

```
IMPPART OBJ(*CURLIB/QRPGSRC) OBJTYPE(*SRC) MBR(BIWKLY)
        PRJ(PAYROLL) GRP(MASTER) TYPE(RPGSRC) PART(EMPMST)
```

If you need to import an object whose name is not valid for an Application Development Manager/400 part, such as an object that has lowercase characters in its name, you must explicitly name the new part. If you use PART(*NAME) here, the IMPPART command fails.

If you are importing more than one source member or object by specifying *ALL on the object (OBJ) or member (MBR) parameters, you cannot explicitly name the parts. Instead, you must let the part name default to the source member name or the object name by specifying PART(*NAME).

Example: The following command imports all the source members in the source file QRPGSRC in the current library and creates parts with the same name as the source members.

```
IMPPART OBJ(*CURLIB/QRPGSRC) OBJTYPE(*SRC) MBR(*ALL)
        PRJ(PAYROLL) GRP(MASTER) TYPE(*OBJTYPE) PART(*NAME)
```

Note that you cannot replace existing parts in the project hierarchy when you use the IMPPART command. If the part already exists in the group you attempt to create it in, or if the part exists in the search path for that group, the IMPPART command fails. You must delete all occurrences of the part in the search path, and try to import the part again.

Optional Parameters

In addition to specifying *from* and *to* criteria when importing objects into a project hierarchy, you can also indicate the following information about the part:

- Language attribute
- Promote code (see “Promote Codes” on page 88 for more information)
- Source file you want the part to be stored in
- Whether you want data copied with physical data files
- Whether you want the existing part replaced
- Part-list part to add the imported part name
- Whether you want to archive the old version of the part being imported
- Whether you want descriptive text associated with the new part

Use the parameters LANG, PRMCODE, SRCFILE, DATA, REPLACE, PARTL, ARCHIVE, and TEXT respectively to indicate this information.

Specify the language attribute for the part on the LANG parameter. The LANG parameter default is *ATTR. If an object is imported, the object type and object attribute are used to determine the Application Development Manager/400 language attribute. If a source file is imported, the member type is used. Refer to Table 4 on page 59 and Table 5 on page 61 to see how AS/400 object types and Application Development Manager/400 part languages relate.

You can specify a language on this parameter, or you can specify LANG(*NONE), which indicates that there is no language attribute. The value *NONE is only valid for parts that do not require a language attribute.

A VRPGTXT or VRPGBIN part has an Application Development Manager/400 part language of VRPG.

Example: If you are importing an RPG source member from a source file, but want to create a part of type CSRC, you must specify a language attribute of C in the IMPPART command as shown in the following command.

```
IMPPART OBJ(*CURLIB/QRPGSRC) OBJTYPE(*SRC) MBR(BIWKLY) PRJ(PAYROLL)
        GRP(MASTER) TYPE(CSRC) PART(BIWKLY) LANG(C)
```

If you are importing more than one source member or object by specifying *ALL on the object (OBJ) or member (MBR) parameters, you must use the default of LANG(*ATTR).

Specify on the SRCFILE parameter where to store the new part. The default is to use the same source file name as that of the imported part by specifying SRCFILE(*FROMFILE). Enter a source file name if you want the part to be stored in a specific source physical file. The special value SRCFILE(*TYPE) is also supported. This indicates that the system-supplied default source file is used. For example, parts of type RPGSRC are stored in the source physical file QRPGSRC. Table 3 on page 49 lists the Application Development Manager/400 part types and their corresponding AS/400 default source files. The SRCFILE parameter is ignored if the type is VRPGTXT.

Specify on the DATA parameter whether to copy data that is associated with a physical data file. The default is to not copy data.

The REPLACE parameter is useful to replace a version of the current part with the version that has been developed outside of the Application Development Manager/400 feature. This function will enable you to import different versions of a part in different group libraries with different promote codes. This was not possible in the previous releases of the Application Development Manager/400 feature.

If REPLACE(*YES) is specified, the current promote code and the source file of the part being replaced will be used, and the SRCFILE parameter will be ignored. If the part that is being imported is not a physical file and it already exists, then:

- The part is checked out to the group where it is to be imported
- The part is replaced with the imported version
- The part is checked in

If a part does not exist in the target group but instead exists in a higher group, then the part is checked out from the higher group during the import operation, and checked back in at the end of the import.

Use the PARTL parameter to specify whether or not you want to add the imported part to the part-list part. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being imported will not be added to a part-list part.

A part-list part must be specified if:

- The group specified on the command was created with PARTLREQ(*YES)
- The type specified is not PARTL

If a part-list part is specified on the IMPPART command, then:

- The part-list part must exist in the default search path starting from the group in which the part is imported.
- The imported part name is added to the specified part-list part, if it does not already exist. (This may happen even if the command failed to import the part).

If the type specified is PARTL, then the PARTL parameter is ignored. In the other cases, the PARTL parameter results in the specified part-list part being updated.

Use the ARCHIVE parameter to specify whether the part being replaced should be archived. This parameter is only meaningful when REPLACE(*YES) is specified and when the part is a source member part. For more information about archiving a part, see “Archiving a Part” on page 94.

On the TEXT parameter, specify descriptive text for the part. You can enter up to 80 characters of text. The description defaults to the text associated with the object or source member if you specify TEXT(*TEXT). The special value *BLANK indicates that the part does not have text associated with it.

Scenario for Importing One Part

This section describes the steps to follow when importing a part.

1. Identify the AS/400 object or member you want to import

   You must determine if the object type is supported by the Application Development Manager/400 feature. See Table 5 on page 61. User-defined part types can also be imported.

2. Import the source member

   Issue the IMPPART command to import the source physical file that contains your source code. The following example imports the source member called PROG1 into a part called BIWKLY in your group DEVELOPER1.

   ```
   IMPPART OBJ(*CURLIB/QRPGSRC) OBJTYPE(*SRC) MBR(PROG1) PRJ(PAYROLL)
           GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
   ```

3. Change the part if necessary

   After importing a part, ensure that all include parts and external references in your source members are not qualified with the library name. If they are, change them to *LIBL, not the name of a specific library. This is necessary to allow the compilers to find the correct version of the part in the project during the build process.

   If you specified a value other than *TYPE on the SRCFILE parameter of the IMPPART command, you should ensure that all include parts are qualified with a file name.

4. Build the part

   Building the part is necessary to ensure that all dependent parts have been imported and are current. It is recommended that you build a part with the following options after you import it.

   PART(*ALL) All the parts imported are built
   TYPE(*ALL) Parts of all types are built

   If you import an AS/400 object only, the build process knows nothing about the part relationships between that object and its corresponding source. You must import the source as well, or the build process issues a warning message when it attempts to build the object. See "Understanding Part Relationships" on page 137 for more information about part relationships and the build process.

5. Verify that the part runs as expected

   See Chapter 12, "Testing and Running an Application."

Importing a Sample Application

This section describes a series of commands to follow to create a new Application Development Manager/400 project hierarchy and import the following hypothetical application from the library called SAMPLE. (The SAMPLE library is not supplied with this product and you need to create it.) It assumes that you are importing mainly source members and not the program objects. Building the application after you import it creates any program objects you require.

Table 6 lists the AS/400 members, with their associated object types and member types, used to create the new Application Development Manager/400 project hierarchy outlined in the steps below.

Table 6. AS/400 Members and Their Associated Object and Member Types

AS/400 Member	Object Type	Member Type
REFMST	*FILE	PF
CTLFIL	*FILE	PF
EMPMST	*FILE	PF
RSNMST	*FILE	PF
TRWEEK	*FILE	PF
TRWEEKL	*FILE	LF
PRG03FM	*FILE	DSPF
PRG06RP	*FILE	PRTF
PROC3	*FILE	CLP
PRG03	*FILE	SQLRPG
PRG03A	*FILE	SQLRPG[1]

Note:

1. This member is actually an include.

Table 7 on page 68 lists an AS/400 object, with its associated object type and object attribute, used to create the new Application Development Manager/400 project hierarchy outlined in the steps below.

Table 7. AS/400 Object and Its Associated Object Type and Object Attribute

AS/400 Object	Object Type	Object Attribute
MESSAGES	*MSGF	*NULL

1. Create the project using the CRTPRJ command. The following command creates the project PAYROLL.

   ```
   CRTPRJ PRJ(PAYROLL) SHORTPRJ(PRL) TEXT('WEEKLY PAYROLL PROCESSING
          APPLICATION')
   ```

   For more information on creating a project, see "Creating a Project (CRTPRJ)."

2. Create the root group in the project PAYROLL. The application is imported into the root group. Other groups in the project hierarchy can be added later. The following command creates the group MASTER in this project.

   ```
   CRTGRP PRJ(PAYROLL) GRP(MASTER) SHORTGRP(MST) PARENT(*NONE)
          TEXT('MASTER GROUP IN PROJECT PAYROLL')
   ```

   For more information on creating a group, see "Creating a Group (CRTGRP)."

3. Identify all the pieces of the application you plan to import, and determine which object types can be imported into the project PAYROLL. See Table 4 on page 59 and Table 5 on page 61 for lists of all the AS/400 object types supported by the Application Development Manager/400 feature.

   Consider using the Work with Members Using PDM display and the Work with Objects Using PDM display to list the AS/400 source file members and objects that are part of your application. You can also use the IM and IO user-defined options from these displays. They are described in "Working with User-Defined Options" on page 237.

4. Use the IMPPART command to import all the source physical files that contain your source code. Use the following command to import all the members within each source file found in the library SAMPLE into the project PAYROLL. In the hypothetical application, note that the member PRG03A is actually an include member.

   ```
   IMPPART OBJ(SAMPLE/*ALL) OBJTYPE(*SRC) PRJ(PAYROLL) GRP(MASTER)
           TYPE(*OBJTYPE) PART(*NAME)
   ```

   If your source code resides in several different libraries, use this command for each library, replacing SAMPLE with the actual library name.

5. Use the IMPPART command to import all the objects that you need for your application. Use the following command to import all the objects in the library SAMPLE. In our hypothetical application, there is only one object being imported and it is called MESSAGES.

   ```
   IMPPART OBJ(SAMPLE/*ALL) OBJTYPE(*NONSRC) PRJ(PAYROLL) GRP(MASTER)
           TYPE(*OBJTYPE) PART(*NAME)
   ```

6. Create a part of type BLDOPT for each part of type CMDSRC that you have imported. The BLDOPT part should have the same name as the part of type CMDSRC, and must contain the create command necessary for this command source. See "Using Build Options" for more information on how to create and manage build options for your application.

7. Change any parts that use library-qualified statements to *LIBL.

8. If after importing you have source parts that must be converted to become include parts, use the CVTPART command. In our example, member PROG3A is actually an include source member. When it was imported, part PROG3A of type RPGSRC was created. This part must be converted to a part of type RPGINC. For information about importing include parts, see Table 4 on page 59.
9. If you want to build the parts you imported, you can specify GROUP(MASTER), TYPE(*ALL), and PART(*ALL) on the BLDPART command to build all the parts you have just imported.
10. Build your application using the BLDPART command with SCOPE(*NORMAL). The following command builds all parts of all types in the group MASTER of the project PAYROLL.

    ```
    BLDPART PRJ(PAYROLL) GRP(MASTER) TYPE(*ALL) PART(*ALL)
            SCOPE(*NORMAL)
    ```

    See Chapter 11, “Building an Application” for more information on how to build an application.
11. Verify that the application runs as expected by testing it. See Chapter 12, “Testing and Running an Application” for information on how to do this.

Chapter 7. Working with Parts

The basic activities you do when working with a part are:

- Check a part out
- Change a part
- Compare parts
- Merge parts
- Check a part in
- Promote a part
- Rename a part
- Delete a part
- Archive a part

You can also perform these activities on the information about a part:

- Change part information
- Print part information
- Retrieve the information about a part
- Convert a part to a part of a different type

Note

All tasks associated with creating, changing, or deleting Application Development Manager/400 information (projects, groups, and parts) should be done using Application Development Manager/400 interfaces. These interfaces include CL commands, and the Programming Development Manager displays (created by the WRKPRJPDM, WRKGRPPDM, and WRKPARTPDM commands). In addition, you can also access Application Development Manager/400 projects, groups and parts from the CODE/400 product and the Application Dictionary Services feature of Application Development ToolSet/400. You may use the CHGPARTINF command to synchronize the part timestamp, allowing the BLDPART command to recognize the parts that have changed.

Checking a Part Out (CHKOUTPART)

Before you can change a part, it must be under your exclusive control. This control is acquired with the Check Out Part (CHKOUTPART) command.

Use the CHKOUTPART command to check a part out to a group to which you have update access. The part is copied to the group specified, from the group it resides in, and a drawdown lock for the part is set at the target group (the final group in a project hierarchy to which a part can be promoted). The access key is set to your user profile. When the access key is set, it prevents others from updating the part. The access key is reset to blank when you use the Check In Part (CHKINPART) command. A **drawdown lock** is associated with the information that is stored about a part. It is the name of the group that contains the lowest occurrence of the part.

If the part does not already exist in the group to which you are checking it out, the CHKOUTPART command copies it from a group that is higher in the project hierarchy. The project hierarchy is scanned, starting with the specified group, for a part with a matching name and type. The part must exist in the default search path for it to be checked out.

If you want to determine whether the part is already checked out to someone else, use the PRTPARTINF command to run a report that shows the person who has the part checked out and the group to which it is checked out. See Figure 30 on page 99 for a sample report.

Required Parameters

Specify the project, group, type, and part on the CHKOUTPART command. Not all types of parts can be checked out.

Optional Parameters

Use the PRMCODE parameter to specify the promote code to be used when determining the promote path for the part. This parameter applies to several commands and is described in “Promote Codes” on page 88. Note that if you have checked a part out with a promote code of *NONE and then decide you want to promote the part, you must delete the part and check it out again with a promote code of *GRP.

PRMCODE(*NONE) must be specified for parts of type PGM, FILE, CLD, CMD, MENU, PNLGRP, MODULE, SRVPGM or user-defined types that are stored in AS/400 objects created by the Build Part (BLDPART) command. Parts of these types can only be promoted when EXTEND(*YES) is specified on the PRMPART command. Note that the parts of type PGM with the languages CBL36 and RPG36 can be checked out with any promote code.

Example: This examples illustrates how the part RPGSRC BIWKLY, which exists in the group TEST, is checked out to the group DEVELOPER1. Figure 22 on page 73 illustrates this.

Using the CHKOUTPART Command

```
CHKOUTPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
           PRMCODE(*GRP)
```

Using the Programming Development Manager utility

Select option 28 (Check out) on the Work with Parts Using PDM display.

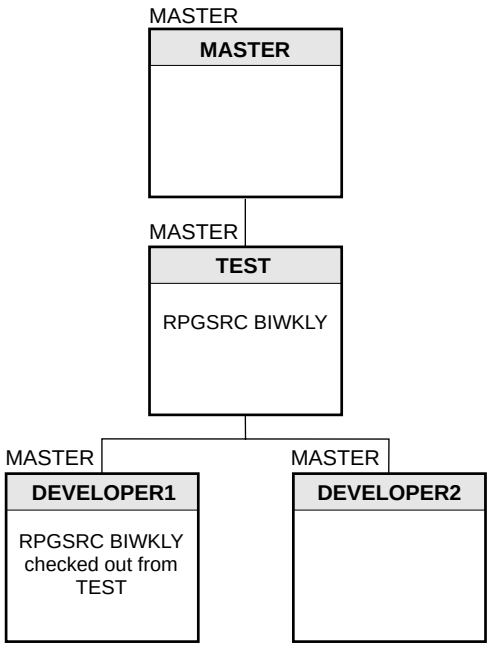

Figure 22. The CHKOUTPART Command and Promote Codes

The promote code for DEVELOPER1 is MASTER so the part RPGSRC BIWKLY is assigned this promote code, meaning that it can eventually be promoted back to the group MASTER.

If you want more information about promote codes, see “Defining One Promote Code for the Project Hierarchy” on page 19 for a description of how the project administrator sets up a promote code. “Promoting a Part (PRMPART)” on page 85 describes the Promote Part (PRMPART) command and the use of the promote code parameter. To see the promote code for a group, issue the Print Project (PRTPRJ) command.

Data Security and Integrity

The drawdown lock for a part prevents that part from being checked out to more than one group. For example, if a part has already been checked out to a development group in another branch of the project hierarchy that uses the same promote code, its drawdown lock has been set for the part at the target group. The target group is the TEST group which prevents you from accessing the part.

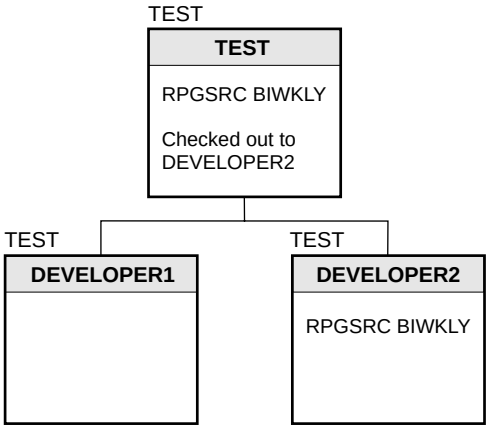

Figure 23. Drawdown Lock Set on a Part

In Figure 23, RPGSRC BIWKLY exists in the group TEST. A developer working in the group DEVELOPER2 has checked the part out to that group, with the intention of promoting it back to the group TEST using the promote code TEST. Thus, if you are working in the group DEVELOPER1, which also has the promote code of TEST, you cannot check this part out. Doing so would mean that two parts with the same name and type would be promoted to the same group. The drawdown lock prevents this from happening.

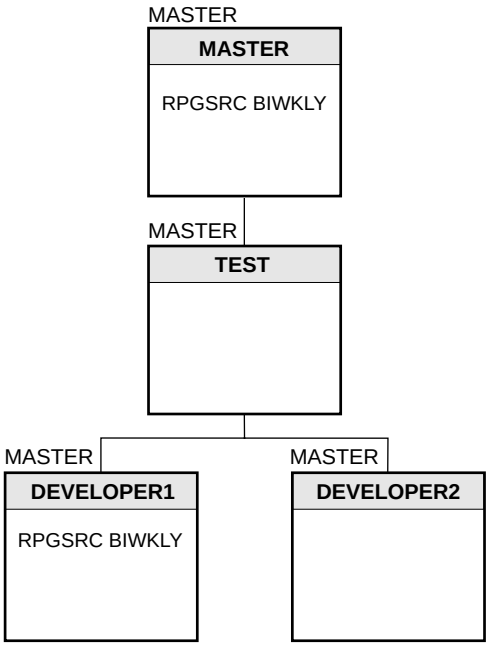

Figure 24. Drawdown Lock Set on a Part

In Figure 24 on page 74, RPGSRC BIWKLY exists in the group MASTER. It is checked out to the group DEVELOPER1 with the intention of promoting it back to the group MASTER. RPGSRC BIWKLY cannot be checked out to the group TEST because the drawdown lock to the group DEVELOPER1 prevents this from happening.

The access key for a part prevents more than one developer from checking out that part. For example, if another developer with access to your group has already checked the part out, a drawdown lock is set for the part, and the access key for the part has been set to the user profile of this person. You can only check out a part if its access key is set to blank.

Changing a Part (CHGPART)

Once you have checked a part out so that it is under your control, you can change it with the Change Part (CHGPART) command. You must have update access to the group containing the part you want to be changed. If the part is already in the specified group, the CHGPART command will first check it out.

Note

If you have *UPDATE access to the group the part resides in, type option 2 (Change) in front of a part on the Work with Parts Using PDM display. The part will be checked out and changed in the group that the part exists and not in the group where you are working.

The CHGPART command runs the AS/400 command that changes the contents or characteristics of a specified part. If the part can be edited, the appropriate editing session is displayed. If it cannot be edited, the appropriate AS/400 change command is displayed to enable you to make the appropriate changes to its characteristics.

Required Parameters

Specify the project, group, type, and part to be changed. Not all types of parts can be changed. See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types that are allowed with the CHGPART command.

Optional Parameters

The CHGCMD parameter determines the AS/400 change command to be called. You can specify *TYPE, STRSDA, STRRLU, or UPDDTA on this parameter.

If you choose the default CHGCMD(*TYPE), the part type and language attribute are used to determine the change command to call. For example, if you change a part of type RPGSRC, you can edit the source that exists in that source part using the SEU editor with its syntax checking capabilities.

If the part cannot be edited, you are prompted for the information you want to change. For example, if you want to change a part of type PGM, you can only change the attributes of the program part, such as the text associated with it.

Table 8 on page 76 shows the commands that are called for a specific part type when you use the default of *TYPE on the CHGCMD parameter.

*Table 8. AS/400 Change Command Called When CHGCMD(*TYPE) is Specified*

Command	Part type to be changed
CHGCMD	CMD
CHGDSPF	FILE with the language attribute of DSPF
CHGDTA	PARTL
CHGDTAARA	DTAARA
CHGICFF	FILE with the language attribute of ICFF
CHGJOBD	For parts of type JOBD
CHGLF	FILE with the language attribute of LF
CHGMNU	For parts of type MENU with the language attribute of UIM
CHGMOD	MODULE
CHGMSGQ	For parts of type MSGQ
CHGOUTQ	For parts of type OUTQ
CHGPF	FILE with the language attributes of PF, VRPGBIN
CHGPGM	PGM
CHGPRTF	FILE with the language attribute of PRTF
CHGSAVF	FILE with the language attribute of SAVF
STRSEU	BLDOPT, BNDSRC, CBL36INC, CBL36SRC, CBLINC, CBLSRC, CBLLEINC, CBLLESRC, CINC, CLDSRC, CLLESRC, CLPSRC, CLSRC, CMDSRC, CSRC, DDSSRC, DSPF36SRC, MNU36SRC, MSGF36SRC, OCL36SRC, PNLINC, PNLSRC, PRDDFN, PRDLOD, REXXSRC, RPG36INC, RPG36SRC, RPGINC, RPGSRC, RPGLEINC, RPGLESRC, RPT36SRC, SCHPTH, SRT36SRC, SYSTEML, TXT36SRC, TXTSRC, and VRPGTXT[1]
WRKBNDDIRE	For parts of type BNDDIR
WRKJOBQ	For parts of type JOBQ
WRKMSGD	MSGF
WRKSCHIDXE	For parts of type SCHIDX

Note:

1. A list of all members in VRPGTXT part is displayed. An individual member can be selected from this list and changed.

For more information on changing a part of type PARTL, refer to "Changing a Part-List Part" on page 104.

For user-defined part types, the *CHG action defined on the ACTION parameter of the CHGADMACN command is started. The actions that can be defined for user-defined types are described in Table 10 on page 131.

Instead of the default CHGCMD(*TYPE), you can specify STRSDA, STRRLU, and UPDDTA if you have the Application Development ToolSet/400 product installed on your AS/400 system.

STRSDA SDA is used to change the part. SDA is used for DDSSRC parts with the language attribute of DSPF. If you have compiled parts using the Screen Design Aid utility called from the CHGPART command,

the build process may not recognize these compilations and will not update the relationships among the parts that have changed.

STRRLU — RLU is used to change the part. RLU is used for DDSSRC parts with the language attribute of PRTF.

UPDDTA — DFU is used to change the part. DFU is used for data files.

Note: Before you use the STRSDA or STRRLU command, you should use the ADDPRJLIBL command to add the project library in which you are working to the AS/400 library list. You can use option 45 (Add project library list) from the Work with Parts Using PDM display or the PDM user-defined option AP instead of the ADDPRJLIBL command. This ensures that SDA and RLU will be able to find any field references in files that are being referenced.

Changing the member source type in SEU or SDA does *not* cause the language attribute of the corresponding part to change accordingly. However, if you change the language attribute of a source part using the CHGPARTINF command, the member source type of the corresponding source member is changed to match it.

Note

Be aware that unpredictable results may occur if parts are changed outside Application Development Manager/400 control. The build process may not recognize that the part has changed and may not recompile or process it. This leads to inconsistencies between parts and their corresponding AS/400 objects and members.

Specifying a PARTL name in the PARTL parameter causes the CHGPART command to automatically add the changed part to the part-list part. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being changed will not be added to a part-list part.

A part-list part must be specified if:

- The group specified on the command was created with PARTLREQ(*YES)
- The type specified is not PARTL

If a part-list part is specified, then:

- The part-list part must exist in the default search path starting from the group in which the parts are changed.
- The changed part name is added to the specified part-list part, if it does not already exist. (This may happen even if the command failed to change the part).

If the type specified is PARTL, then the PARTL parameter is ignored. The PARTL parameter results in the specified part-list part being updated.

Use the ARCHIVE parameter to specify whether the part being replaced should be archived. For more information about archiving a part, see “Archiving a Part” on page 94.

Example: The following command changes the part RPGSRC BIWKLY in the group DEVELOPER1.

Using the CHGPART Command

```
CHGPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
        CHGCMD(*TYPE)
```

Using the Programming Development Manager utility

Select option 2 (Change) on the Work with Parts Using PDM display.

CHGCMD(*TYPE) specifies that the part type and language attribute are to determine the AS/400 change command that is called to change the part. In this example, the part would be placed in an SEU editing session.

You may use your own editor to change the part. You can write a CL or REXX program that retrieves the name of the AS/400 source member that corresponds to the name of the Application Development Manager/400 part, and then use this information to call your editor. See “Retrieving Part Information (RTVPARTINF)” on page 100 for an explanation of how to do this.

When you are finished editing the part using your own editor, you should use the CHKINPART command to check the part back in. The part's timestamp and the user who last changed the part are updated. This way, the next time you issue the BLDPART command, the part will be built.

Converting a Part's Type and Language (CVTPART)

The Convert Part (CVTPART) command converts a part that is a source member to a part of a different type within the same project and group. After importing, it is useful to convert a part of type RPGSRC to an include part with a part type of RPGINC, or to convert a part of type CSRC to an include part with a part type of CINC.

Groups in the project hierarchy are not searched for the part to be converted if the part is not located in the group specified on the CVTPART command.

Note: ILE RPG/400 provides its own conversion command: the Convert RPG Source (CVTRPGSRC) command. The command converts source members containing RPG/400 source to members containing ILE RPG/400. The source file containing the resulting members must have a record length of 112 bytes or more. Due to this reason, the RPGLESRC and RPGLEINC parts must reside in a source file that has a record length of at least 112 bytes. In addition, the restricted values listed in Table 9 on page 79 must be imposed.

In general, the CVTPART command creates a new part without changing the part. However, if the original part is a source member and you specify that the source file for the part being converted should be the same as the one used to store the original part, the original part is deleted. The new part and type represents the original member. For example, let us examine a source file called QRPGSRC and a member part called A with a part type of RPGSRC. When you convert this source file to part type RPGINC without changing the part name, the CVTPART command will delete the original part RPGSRC A and copy it to a new part called RPGINC A.

Rules: The following rules apply:

1. The TOTYPE name must be different from the FROMTYPE name.
2. Parts of type FILE cannot be converted.
3. Parts of type PGM or SRVPGM cannot be converted to parts of type of PGM or SRVPGM.
4. Parts that are source members can only be converted to other parts that are also source members.
5. Parts of type TXTSRC can be converted to parts of any type that are also source members.
6. Any type that is stored in source members can be converted to parts of type TXTSRC.

Restrictions: In addition to the rules mentioned above, the combinations of FROMTYPEs and TOTYPEs listed in the table below can be used on the CVTPART command.

Table 9. List of Restricted Values for FROMTYPEs and TOTYPEs for the CVTPART Command

Fromtype	Totype
CLD	CLDSRC
PGM with language CLLE	CLLESRC
PGM with language CLP	CLPSRC
RPGSRC	RPGINC, RPGLEINC, RPGLESRC
RPGINC	RPGSRC, RPGLEINC, RPGLESRC
RPGLESRC	RPGLEINC
RPGLEINC	RPGLESRC

Note: The build information is not copied to the new part when a part is converted.

This CVTPART function is also available by using option 50 (Convert part) on the Work with Parts Using PDM display.

Example: Take the following steps to convert a part of type TXTSRC to a part of type PNLSRC and language PNLGRP.

1. Convert the part type to PNLSRC using the CVTPART command. This will set its language to MENU.
2. Change its language to PNLGRP using the CHGPARTINF command.

Now by default, when you convert a part of type TXTSRC to a part of type CSRC or CINC, the part language is set to CLE. So you can also use the above steps to convert a part of type TXTSRC to a:

- Part of type CSRC with language C
- Part of type CINC with language C
- Part of type CSRC with language SQLC
- Part of type CINC with language SQLC

Required Parameters

Specify the project, group, and type representing the part whose type you want to convert. If you want to convert all parts, use the default PART(*ALL). You must also specify the new type.

Optional Parameters

You can specify:

- A name for the part being converted. The default is TOPART(*FROMPART). The name of the part being converted remains the same.
- A promote code. The default is PRMCODE(*GRP). The promote code for the group is used.
- The name of the source file where the part should be stored as a source member. The default is SRCFILE(*TYPE). The AS/400 default source file is used. If you specify SRCFILE(*SAME), the name of the source file that the original part uses is also used. The value *TYPE is assumed if the following part types are being converted: PGM to CLPSRC and CLD to CLDSRC. See Table 3 on page 49 for a list of part types and their corresponding AS/400 default source files. The SRCFILE parameter also applies to user-defined types stored in AS/400 source file members.
- The name of the part-list part to which the converted parts are added. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being converted will not be added to a part-list part.

  A part-list part must be specified if:

  – The group specified on the command was created with PARTLREQ(*YES)
  – The type specified is not PARTL

  If a part-list part is specified on the CVTPART command, then:

  – The part-list part must exist in the default search path starting from the group in which the parts are converted.
  – The names of the converted parts are added to the part-list part, if they do not already exist. (This may happen even if the command failed to convert the part).

  If the type specified is PARTL, then the PARTL parameter is ignored. In the other cases, the PARTL parameter results in the specified part-list part being updated.

Comparing Parts (CMPPART)

The Compare Part (CMPPART) command compares two parts and documents the differences between them in a report. This can be particularly useful when you must migrate a fix to a follow-on version of a part.

The following types of parts may be compared:

- Parts stored in source file members

- Parts of part type FILE and language PF (physical file), and VRPGBIN
- Parts of part type PARTL (part lists)
- Parts of part type VRPGTXT

To compare the individual native source members of the VRPGTXT part, use the CMPPFM command on the native source physical file storing the part.

Required Parameters

Specify the project, group, type, and part representing the new part being compared. The new part must exist in the project and group. The default group hierarchy is not searched for any parts not found in this group. If you want to compare all parts, use NEWTYPE(*ALL) and NEWPART(*ALL). You must also specify the OLDTYPE(*NEWTYPE) and OLDPART(*NEWPART).

Optional Parameters

Specify the group, type, and part of the old part being compared. The default hierarchy is searched for any parts not found in this group.

To specify the type of compare to be performed, use the CMPTYPE parameter. Use CMPTYPE(*LINE) to compare and identify inserted, deleted and updated lines; CMPTYPE(*FILE) to see whether the contents of the two files are different or the same; and CMPTYPE(*WORD) to compare and identify added, deleted and updated words.

The RPTTYPE command is used to specify the type of the report. The default, RPTTYPE(*DIFF), lists only the differences between the members. Use RPTTYPE(*SUMMARY) to list a summary of the results of the comparison, without showing the detailed differences; RPTTYPE(*CHANGE) to provide the same information as the *DIFF report type, with ten lines before and after the differences; and RPTTYPE(*DETAIL) to list the entire new file, indicate the differences, and provide a summary of the results.

Use the OUTPUT parameter to indicate where the output should go. The default is OUTPUT(*). The report is displayed on your workstation. If the command is run in batch, then the output will be spooled to the output queue of the print device. OUTPUT(*PRINT) spools the output to the print device. OUTPUT(*OUTFILE) directs the output to the output file.

The OPTION parameter describes how you want the comparison to be carried out. Up to 12 option choices may be specified at any one time. For more information about the choices for the OPTION parameter, refer to the CMPPFM command in *ADTS/400: File Compare and Merge Utility*.

Specify the library name and the name of the file on the STMTFILE parameter which contains the member with comparison directive statements. For more information about the comparison directive statements, refer to *ADTS/400: File Compare and Merge Utility*.

The STMTMBR parameters specifies the member that contains comparison statements.

Example: The following command compares a part RPGSRC BIWKLY in the group DEVELOPER1 with a part RPGSRC BIWEEK in the same group. The report is spooled to a print device.

Using the CMPPART command

```
CMPPART PRJ(PAYROLL) NEWGRP(DEVELOPER1) NEWTYPE(RPGSRC) NEWPART(BIWKLY)
        OLDGRP(DEVELOPER1) OLDTYPE(RPGSRC) OLDPART(BIWEEK)
        OUTPUT(*PRINT)
```

Using the Programming Development Manager utility

Select option 54 (Compare part) on the Work with Parts Using PDM display.

Merging Parts (MRGPART)

The Merge Part (MRGPART) command merges specific parts or all the parts in the default search path from a specified target group. Only parts stored in source files may be merged. This 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.

The MRGPART command compares each target part and maintenance part with its corresponding root part. The results of these comparisons are used to determine the updates that have occurred.

To merge individual native source members found inside of a VRPGTXT part, use the MRGSRC command on the native source physical file storing the part.

This command uses the MRGSRC command. For more information on the MRGSRC command, see the *ADTS/400: File Compare and Merge Utility* book.

Required Parameters

The target project, target group, target type, and target part name specify where the part will be placed after the merge. The **target** is a source part which may also contain updates that are merged with the maintenance part. The result from the merge is placed into the target part. You must have update access to the group in which the target part is being merged. The target part will be checked out to the target group. See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types that are allowed with the MRGPART command.

The default hierarchy will be searched to obtain the list of parts to be merged.

If TGTTYPE(*ALL) is specified then MAINTTYPE(*TARGET) must be specified; if TGTPART(*ALL) is specified then MAINTPART(*TARGET) must be specified.

If TGTTYPE(*ALL) is specified then ROOTTYPE(*TARGET) must be specified; if TGTPART(*ALL) is specified then ROOTPART(*TARGET) must be specified.

Optional Parameters

The maintenance project, maintenance group, maintenance type, and maintenance part name specify where the changed parts that are to be merged are kept. The **maintenance** is the source part containing the updates to be merged into the target part.

The root project, root group, root type, and root part name specify where the original parts that are to be merged are kept. The **root** is the original version of the source part, on which both sets of updates are based.

Specify the part-list part on the PARTL parameter where you may add the target parts. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being merged will not be added to a part-list part.

A part-list part must be specified if:

- The group specified on the command was created with PARTLREQ(*YES)
- The type specified is not PARTL

If a part-list part is specified on the MRGPART command, then:

- The part-list part must exist in the default search path starting from the target group in which the parts are merged.
- The names of the merged parts are added to the part-list part, if they do not already exist.

If the type specified is PARTL, then the PARTL parameter is ignored. In the other cases, the PARTL parameter results in the specified part-list part being updated.

The PARTL parameter is also not required if RPTONLY(*YES) was specified.

Use the SELECT parameter to specify whether you want the Split Merge display shown so that you may select the individual changes in the maintenance part that are to be merged into the target part. If the MRGPART command is run in batch, the SELECT is ignored and the Split Merge display is not shown.

Use the RPTONLY parameter to indicate whether you want to perform the merge or produce a report only. This parameter is only prompted if SELECT(*NO) was specified on the MRGPART command. If SELECT(*YES) was specified, then RPTONLY(*NO) must be specified.

Example: The following command merges the original version 1 part, the changed version 1 part, with the new version 2 part.

Using the MRGPART command

```
MRGPART PRJ(PAYROLL) TGTGRP(V2) TGTTYPE(RPGSRC) TGTPART(BIWKLY)
        MAINTGRP(V1CHANGE) ROOTGRP(V1)
```

Using the Programming Development Manager utility

Select option 55 (Merge part) on the Work with Parts Using PDM display.

Merging a Sample Application

Consider that you have received an application containing three parts, PART A, PART B, and PART C, from your vendor. The R1ROOT group contains the vendor supplied first release of the application. You have changed PART A and PART B and have created a new part, PART D. The R1MAINT group contains the first release parts changed and added by you. Some time has passed and now your vendor has sent you a second release of the application with PART A and PART C

updated and a new part, PART E. The R2ROOT group contains the parts containing the second release of the application. Your project hierarchy is shown in the Before portion of Figure 25.

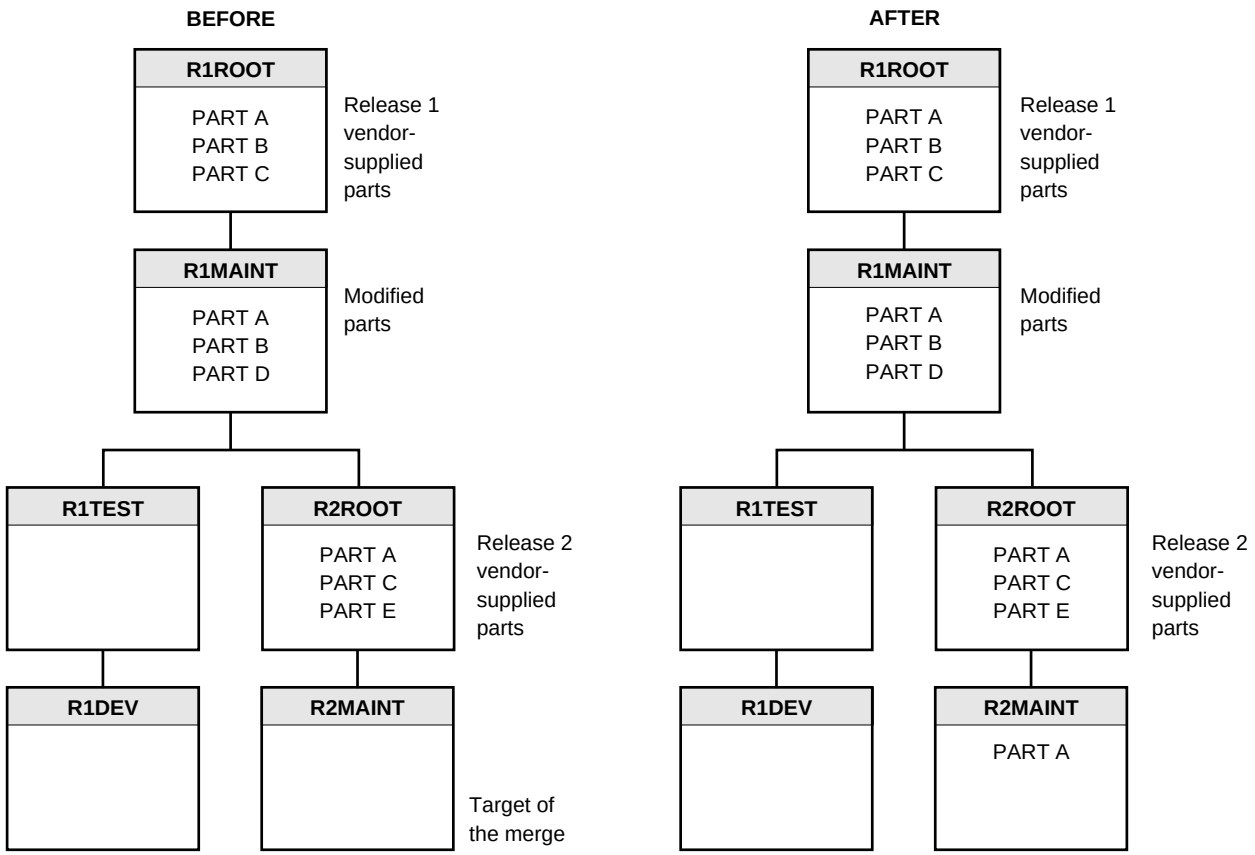

Figure 25. Example Showing the Project Hierarchy Before and After the Merge

Use the following command to merge the vendor's changes in R2ROOT and your changes in R1MAINT, and place the result in the R2MAINT group.

```
MRGPART PRJ(ACCTA) TGTGRP(R2MAINT) TGTTYPE(*ALL) TGTPART(*ALL)
        MAINTGRP(R1MAINT) ROOTGRP(R1ROOT)
```

Your project hierarchy after the merge is shown in the After portion of the Figure 25. Note that PART A is the only part which gets checked out and merged in the target group R2MAINT because two different versions of this part occur in the groups R1MAINT and R2ROOT.

After you performed the merge, you might have decided to change part PART C in the R1MAINT group. If so, after making your change, you would merge it with the copy of the part PART C sent by your vendor for the second release using the merge procedure mentioned above.

To improve performance, use the merge procedure mentioned above, except specify TGTPART(C) on the MRGPART command.

Checking a Part In (CHKINPART)

The Check In Part (CHKINPART) command releases a part that has been checked out by resetting its access key to blank. This allows the part to be used by others sharing the same development group. Checking a part in does not release its drawdown lock, so developers in other groups still cannot access it. The drawdown lock is only released when the part is promoted to the target group. The CHKINPART command does not move parts up to the next group in the project hierarchy.

The CHKINPART command also ensures that the timestamp of a part is updated to match the timestamp of the corresponding AS/400 object or member. This ensures that the build process recognizes the part has changed, and recompiles the part the next time the BLDPART command is issued.

The user profile of the person checking a part in must be the same as the profile that checked it out; however, the project administrator can check any part in by resetting its access key to blank with the CHGPARTINF command. (Only the project administrator has the authority to change the access key with the CHGPARTINF command.)

Required Parameters

Specify the project, group, type, and part. See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types that are allowed with the CHKINPART command.

If you want to check several parts in at the same time, use option *ALL on the TYPE or PART parameter. This checks in all parts of a certain type, or all parts that are checked out to you.

Example: This example illustrates how to check in the part RPGSRC BIWKLY which is in the group DEVELOPER1.

Using the CHKINPART command

```
CHKINPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
```

Using the Programming Development Manager utility

Select option 29 (Check in) on the Work with Parts Using PDM display.

Promoting a Part (PRMPART)

The Promote Part (PRMPART) command moves a specified part one group at a time, from the group the part is in, to its parent group. You can only promote parts from groups to which you have update access. The access key of the part being promoted must either be blank or set to your user profile. That is, the part must be checked out to you.

If the part to be promoted is checked out, it is automatically checked in first before being promoted. When you check a part out, the access key for the part is set to your user profile so that you have exclusive update access to the part. When you promote the part, the access key is reset to blank so that other developers can use it.

VRPGTXT and its associated VRPGBIN parts should be promoted together using the PRMPART command. This maintains the integrity between VRPG Client parts since the VRPGTXT part may contain some output of its associated VRPGBIN part.

Required Parameters

Specify the project, group, type, and part. Part types that are allowed to be promoted will be promoted as long as they are not checked out to someone else. See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types that are allowed with the PRMPART command.

Optional Parameters

Parts that are generated by building source parts are output parts and can only be promoted if you specify EXTEND(*YES) on the PRMPART command. EXTEND(*YES) is the only means by which parts with a promote code of *NONE can be promoted. The PRMPART command will delete the PGM and PF parts in the lower group after a successful extended promote. If logical files are based on any physical file part being promoted, these parts will not be deleted from the lower group after the extended promote.

If you want to promote several parts at the same time, you can do this using the TYPE or PART parameter with the *ALL option. This allows you to promote all parts of a certain type or all parts of all types not checked out to someone else. Another way to promote several parts at once is to create a part-list part (part of type PARTL) and use the PARTLOPT parameter to promote the part-list part. If you specify PARTLOPT(*LIST), for example, each part listed inside the part-list part is promoted.

The PARTL parameter is where you specify if you want to add the promoted part to the part-list part. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being promoted will not be added to a part-list part.

A part-list part must be specified if the parent group specified on the command was created with PARTLREQ(*YES).

On the PRMPART command, you can specify TYPE(PARTL) with the PARTL parameter because this command can expand the part-list part and process each part from the part-list separately.

If a part-list part is specified on the PARTL parameter, then:

- The part-list part must exist in the default search path starting from the group from which the parts are promoted.
- The promoted parts are added to the specified part-list, if they do not already exist. (This may happen even if the command failed to promote the part.)

Use the ARCHIVE parameter to specify whether the part being replaced should be archived. For more information about archiving a part, see “Archiving a Part” on page 94.

Example: This example illustrates how to promote the part RPGSRC BIWKLY from the group DEVELOPER1 to the next group in the project hierarchy, the group TEST.

Using the PRMPART command

```
PRMPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
```

Using the Programming Development Manager utility

Select option 30 (Promote) on the Work with Parts Using PDM display.

Promoting Output Parts Created by the Build Process

Use the EXTEND parameter if you want to promote the output parts of the build process along with the associated source parts.

The default is EXTEND(*NO). Only the part you specify on the PRMPART command is promoted. If you specify EXTEND(*YES), the specified part is promoted along with the output part created when the part was last built. In addition, the service data area of the object information repository (OIR), where the source file, library, and member names of the source used to compile the object are stored, is updated to reflect the new library name. Parts created by the build process have a promote code of *NONE. EXTEND(*YES) is the only means by which you can promote parts that have a promote code of *NONE.

The following rules apply when you promote a part that has a promote code of *NONE:

- If the same part already exists with a promote code other than *NONE in the group to which you are promoting, you cannot promote the part with the promote code *NONE.
- All related physical files and logical files must be located in the same group. A physical file must not exist in one group and its associated logical file in another.
- The part continues to have a promote code of *NONE after you promote it. The project administrator or any user who has *UPDATE access to the group to which the part was promoted should run the BLDPART command on all the parts to ensure they are up-to-date. Those parts already up-to-date are not compiled.

Promoting a Part-List Part

If you are promoting a part-list part, the PRMPART command allows you to specify how you want to promote it. The PRMPART command processes a part-list part as follows:

1. The default search path is used to search for the part-list part. If the part-list part is not found, the PRMPART command fails. If PARTLOPT(*PART) is specified, then the PARTL part must exist in the specified group.

2. If you specify PARTLOPT(*LIST) on the PRMPART command, each part in the part-list part is promoted. If you specify PARTLOPT(*PART), the part-list part is promoted like any other part. If you specify PARTLOPT(*BOTH), both the part-list part and the parts listed inside it are promoted. You can also specify generic part names on the PRMPART command. See Chapter 8, “Part-List Parts, Reason Control, and Change Tracking” for more information.

If the command you are running ends abnormally, it is possible that some parts in it are not processed. To ensure that all the parts are processed, you should look for any error messages and, if necessary, run the command again.

Example: The following command promotes the part-list part PART001 from the group DEVELOPER1 to the next group in the project hierarchy, the group TEST:

```
PRMPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(PARTL) PART(PART001)
        PARTLOPT(*LIST)
```

Promote Codes

This section describes the promote code (PRMCODE) parameter as it applies to the CHKOUTPART, CPYPART, CRTPART, IMPPART and RNMPART commands. The project administrator defines the promote code for a particular group at the time the group is created.

Typically you use only one promote code when working with parts. However, the project administrator might have defined several promote paths, and therefore several promote codes, within one project hierarchy. This may be the case if your organization must maintain an existing version of an application while updating another version of the same application. See “Two Versions of One Part” on page 23 for an example of this situation.

The PRMCODE parameter specifies the promote code to be used to determine the promote path in the project hierarchy for a part. There are two values that can be specified.

*GRP The promote code for the group specified on the GRP parameter (TOGRP parameter for the CPYPART command) is assigned to the part.

If *GRP is specified for the CHKOUTPART, CPYPART, or IMPPART command, and the part has a part type of PGM, FILE, CLD, CMD, MENU, PNLGRP, MODULE, SRVPGM, or is a user-defined type stored in AS/400 objects created by the BLDPART command, it is processed with PRMCODE(*NONE). Parts of these types cannot be promoted unless you use the EXTEND parameter. EXTEND(*YES) is the only means by which parts with a promote code of *NONE can be promoted. Parts of types that can be created by the BLDPART command can be promoted when the parts created by the BLDPART command are parts that are stored in source members. Note that the parts of type PGM with the languages CBL36 and RPG36 can have a promote code of *GRP.

*NONE No promote code is assigned to the part. The part cannot be promoted.

Using the CHKOUTPART, CHKINPART, and PRMPART Commands

Figure 26 and Figure 27 on page 90 use the sample project hierarchy to illustrate what happens to a part before and after you use the CHKOUTPART, CHKINPART, and PRMPART commands.

In the Before portion of Figure 26, an RPGSRC part called BIWKLY exists only in the group TEST. You want to make changes to the part. You use the CHKOUTPART command to check it out to your development group called DEVELOPER1. This is illustrated in the After portion of the figure. BIWKLY is copied to your group and a drawdown lock is set for the part. The access key is set to your user profile, preventing others from updating the part.

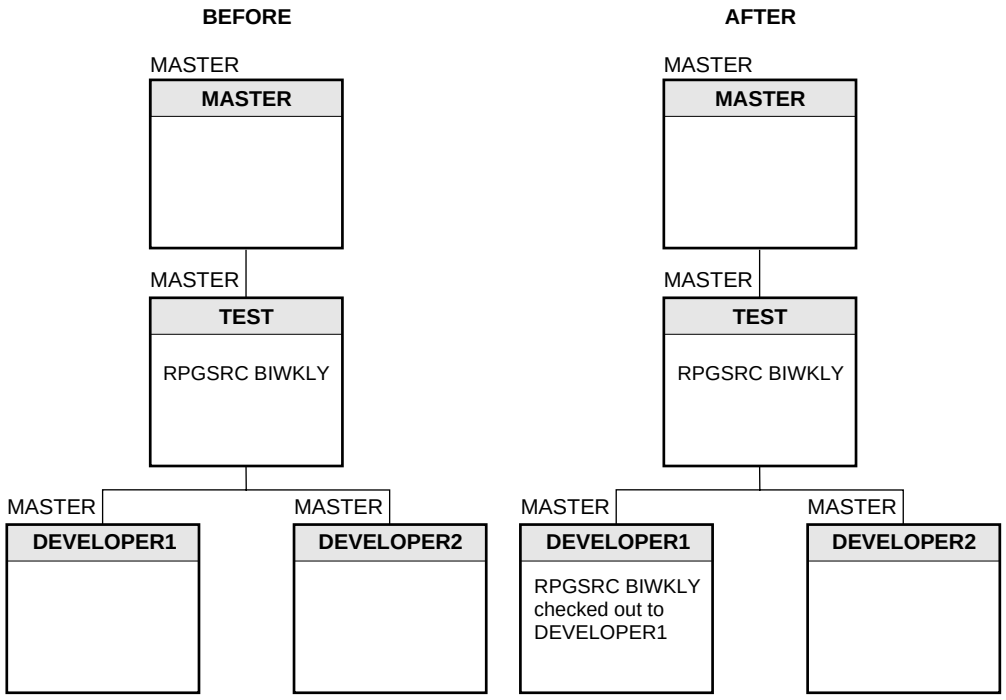

Figure 26. CHKOUTPART Command

Now that BIWKLY is checked out to your group, you can make changes to it. When you finish making changes, you use the CHKINPART command to check the part back in. The part is still in DEVELOPER1 as illustrated in the Before portion of Figure 27 on page 90. If this group is shared by other developers, one of them can now check out the part and change it.

The developer in the group DEVELOPER2 cannot check BIWKLY out, because the part has already been checked out to the group DEVELOPER1 (regardless of whether the part is checked out to the developer in the group DEVELOPER1). The part is waiting to be promoted to the group TEST and is not available to the group DEVELOPER2.

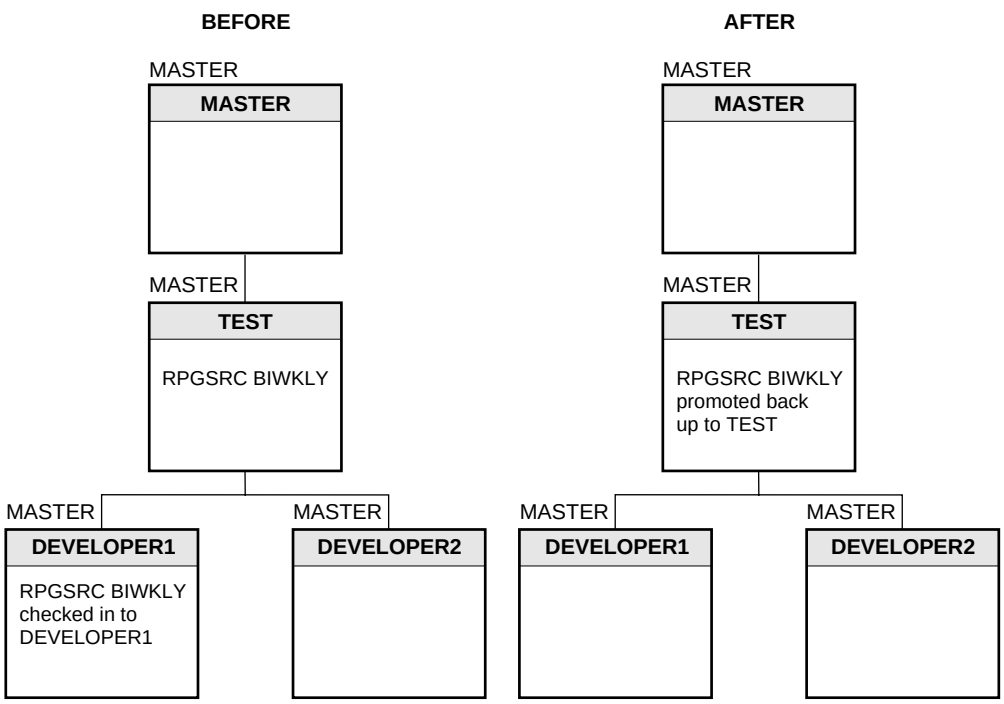

Figure 27. CHKINPART and PRMPART Commands

You now use the PRMPART command to promote it back to the group TEST. The After portion of Figure 27 shows how the PRMPART command moves the part up one level to the group TEST. If the part is still checked out to you, the PRMPART command checks the part back in to DEVELOPER1 first, before promoting it back up to the group TEST. Promoting BIWKLY to the group TEST replaces the old copy of that part in TEST with the new part coming from DEVELOPER1.

The following commands produce the actions shown in Figure 26 on page 89 and Figure 27:

1. To check out the part RPGSRC BIWKLY, enter:

   ```
   CHKOUTPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
              PRMCODE(*GRP)
   ```

2. To change the part RPGSRC BIWKLY, enter:

   ```
   CHGPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
           CHGCMD(*TYPE)
   ```

3. To check the part RPGSRC BIWKLY back into the group DEVELOPER1, enter:

   ```
   CHKINPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
   ```

4. To promote the part RPGSRC BIWKLY back up to the group TEST, enter:

   ```
   PRMPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
   ```

Renaming a Part (RNMPART)

Use the Rename Part (RNMPART) command to rename parts from one name to another name within the same project and group.

A part of a given name and type can only be renamed if a part of the new name and of the same type does not exist in the specified group. If a part of the same name and type already exists in the group, a message is issued and the RNMPART command fails.

A renamed part will not maintain the build information that may have existed for the old part. A part is renamed only in the specified group and not in the entire project.

You must have update access to the group containing the part being renamed.

If the part was checked out to you before you renamed it, then the part will be renamed but it will not be checked out to you. If the part has been checked out to another user, a message will be issued and the RNMPART command fails.

If you want to change attributes of the copied part, such as its language or text, use the Change Part Information (CHGPARTINF) command after the rename is complete.

Parts of type VRPGTXT or VRPGBIN cannot be renamed using this command.

Required Parameters

Specify the project, group, type, and part name of the part being added and the new name of the part. You must have update access to the group in which the part is being renamed. Any type of part may be renamed using the RNMPART command.

To rename a part of a user-defined type, *CPY and *DLT actions must have been defined for the part type.

The NEWPART parameter lets you specify the new name of the part. This name must be different from the name specified on the PART parameter.

Optional Parameters

Use the PRMCODE parameter to specify whether the new part is to be promoted. This parameter applies to several commands and is described in the section called “Promote Codes” on page 88.

The PARTL parameter is where you may add the renamed part to the part-list part. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being renamed will not be added to a part-list part.

A part-list part must be specified if:

- The group specified on the command was created with PARTLREQ(*YES)
- The type specified is not PARTL

If the type specified is PARTL, then the PARTL parameter is ignored. In the other cases, the PARTL parameter results in the specified part-list part being updated.

Example: The following command renames the part RPGSRC BIWKLY to RPGSRC BIWEEK in group TEST of PAYROLL project.

Using the RNMPART command

```
RNMPART PRJ(PAYROLL) GRP(TEST) TYPE(RPGSRC) PART(BIWKLY)
        NEWPART(BIWEEK) PRMCODE(*GRP) PARTL(*NONE)
```

Using the Programming Development Manager utility

Select option 7 (Rename) on the Work with Parts Using PDM display.

Deleting a Part (DLTPART)

Use the Delete Part (DLTPART) command to delete a part from a group. You must have update access to the group and the part must not be checked out to someone else.

Required Parameters

Specify the project, group, type, and part. See Appendix B, “Part Types and Their Relationship to Commands” on page 247 for a list of part types that can be deleted by the DLTPART command.

Use TYPE(*ALL) to delete parts of all types, or PART(*ALL) to delete all parts.

You must delete logical file parts before deleting the associated physical file part.

If the part being deleted is not checked out to somebody else, the part, and any information the Application Development Manager/400 feature saves about the part, is deleted. If the part is checked out, it is not deleted.

Optional Parameters

The PARTL parameter is where you specify if you want to remove the deleted part from the part-list part. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part being deleted will not be added to a part-list part.

Note that the PARTL parameter is optional on the DLTPART command, even though the group specified on the command was created with PARTLREQ(*YES).

If a part-list part is specified and the part to be deleted is not of type PARTL, then:

- The part-list part must exist in the default search path starting from the group in which the part is deleted.
- The deleted parts are removed from the part-list part. (This may happen even if the command failed to delete the part.)

If the type specified is PARTL, then the PARTL parameter is ignored. In the other cases, the PARTL parameter results in the specified part-list part being updated.

Use the DLTARCHIVE parameter to specify whether the archived versions of the part should be deleted with the part. When all the archived members are deleted, the source file is also deleted if there are no other members in the file.

Example: This example illustrates how to delete the part RPGSRC BIWKLY from the group DEVELOPER1.

Using the DLTPART Command

```
DLTPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
```

Using the Programming Development Manager utility

Select option 4 (Delete) on the Work with Parts Using PDM display.

How the Drawdown Lock Changes

When you delete a part and the same part exists in more than one group, the drawdown lock changes to point to the group to which the part is currently checked out. Consider Figure 28:

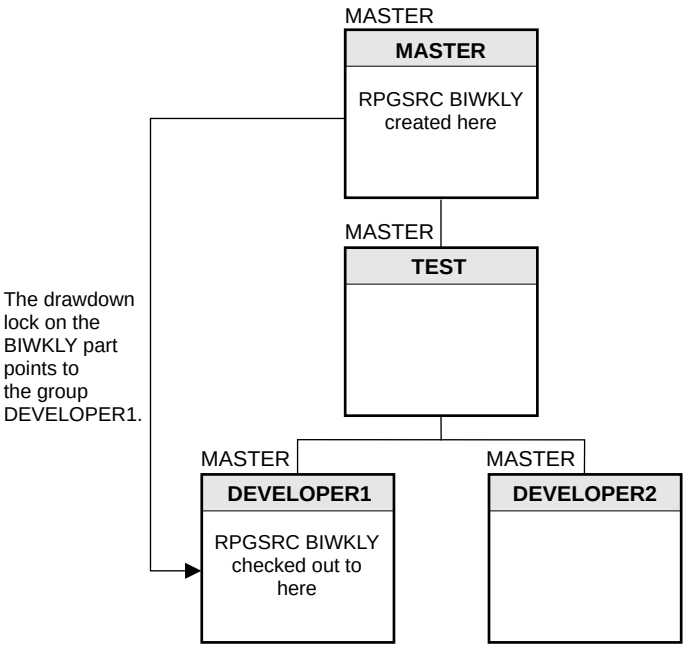

Figure 28. The DLTPART Command and the Drawdown Lock

Suppose that you promote the part from the group DEVELOPER1 to the group TEST, and then you check it out again to the group DEVELOPER1. Copies of the part BIWKLY now exist in all three groups: DEVELOPER1, TEST, and MASTER. If you delete the copy of the part in your group DEVELOPER1, the drawdown lock points to the group TEST. This is illustrated in Figure 29 on page 94.

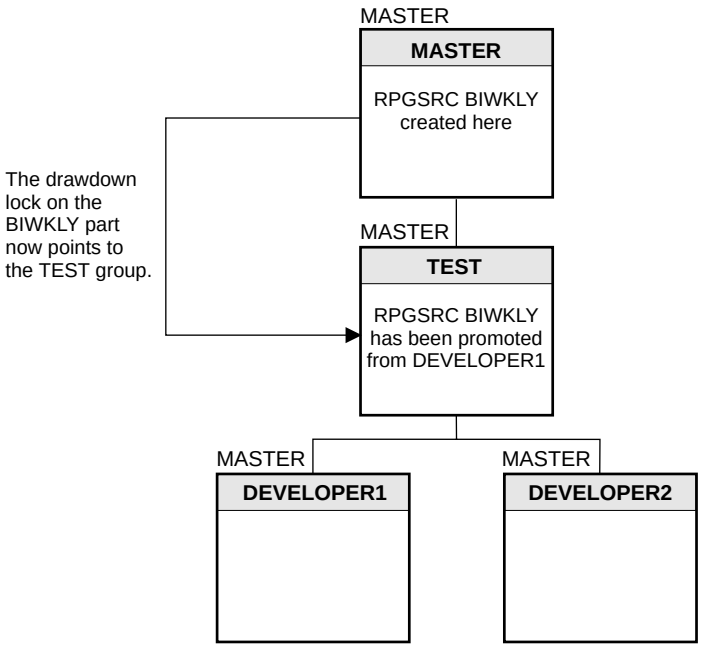

Figure 29. A Change in the Drawdown Lock

Project Administrator Intervention

Since one part can exist in more than one group, deleting it entirely from the project hierarchy may require intervention from the project administrator. This situation arises if you delete a part, but a copy of the part still exists in another group to which you do not have update access. The administrator must delete that version of the part.

Archiving a Part

In an Application Development Manager/400 environment, with the contents of parts constantly changing, you might want to keep back-level versions of these parts. Through the use of the ARCHIVE parameter, the following commands can now automatically **archive** up to five versions of a source part in a given group.

CHGPART
IMPPART
PRMPART

A **group archive library** contains the back-level versions of parts, and is created by the above commands for the group the part resides in. This library can contain source files with their names the same as their part names, and each containing up to five members of back-level versions. The member name is "archive*n*", where n=1 is the most recent version and n=5 is the oldest version.

If ARCHIVE(*YES) is specified:

- A group archive library is created if it does not already exist. The name of the group archive library is xxxx_yyyyy, where xxxx is the short project name and yyyyy is the short group name. The group archive library has the same authority as the group library.

- The archive library for the group containing the part being replaced is checked for a source file with the name of the part being replaced. A source file with that name is created if one does not exist.
- If the source file contains a member called "archive5", then that member is deleted.
- The remaining members are renamed, so that the number in the member name increases by one. For example, "archive4" becomes "archive5".
- The member being replaced will be copied into member "archive1".

Note: It is possible to have two parts with the same name (but with different types) in the same group. Such parts will be archived in the same source file, so one archived part could overwrite the other. It is, therefore, advisable to use unique part names, even for parts of different types, if they need to be archived.

To retrieve or **roll back** an archived part, use the IMPPART command. Specify the name of the group archive library on the LIBRARY prompt and the name of the archived member on the MEMBER prompt. Remember to specify the member name in lower case with double quotation marks around it. You may use REPLACE(*YES) to roll back the changes into a part.

If you are working with the Work With Parts Using PDM display or the Work With Parts in Part List Using PDM display, you can type option 52 (Work with archived members) beside the part to take you to the Work With Members Using PDM display listing archived members of the part. Then you can use the user-defined option IM (Import Part) to roll back the changes. For more information, see the “Working with Archived Members” on page 232 and “User-Defined Options” on page 236.

The DLTARCHIVE parameter on the DLTPART command allows you to specify whether the archived versions of the part should be deleted along with the part itself. When all the archived members are deleted, the source file is also deleted if there are no other members in the file.

Information Associated with Parts

Each part has information associated with it that you can change and print. Use the Change Part Information (CHGPARTINF), Print Part Information (PRTPARTINF), and Retrieve Part Information (RTVPARTINF) commands to work with part information.

Changing Part Information (CHGPARTINF)

The Change Part Information (CHGPARTINF) command can be used to change the following information about a part, one part at a time:

- Language of a part. Table 2 on page 45 lists part types and their associated language.
- Access key (user profile) of the person holding the part (only a project administrator can change this).
- Text description for a part.

If you are using the CHGPARTINF command to change the language of a part, the part timestamp is set to the current time, and the user profile of the person who last changed the part is set to the current user.

If you are not using the CHGPARTINF command to change the language of a part, CHGPARTINF ensures that the part timestamp is made equal to the last changed timestamp of the part's object or member. This happens even if you have specified *SAME for the language, access key, and text description.

The Build Part command (BLDPART) will not build parts of type CSRC, CINC and PGM with language C and SQLC. In order to retain the data integrity, these parts will not be automatically converted to language CLE or SQLCLE if the parts of these types already exist. In order to change the language to CLE or SQLCLE, you can use the CHGPARTINF command.

Required Parameters

Specify the project, group, type, and part whose information you want to change. See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types for which part information can be changed using the CHGPARTINF command.

Optional Parameters

If you change the language attribute of a part, and the part is one of the types listed in Table 4 on page 59 or a user-defined type, the source type of the corresponding AS/400 source member that contains the part is changed accordingly. User-defined source member types are also changed.

The default for the language parameter is LANG(*SAME). It specifies that the current language is not changed. *SAME will be replaced with the current language when all the required parameters on the CHGPARTINF command are filled in. Use LANG(*NONE) if you do not want any language to be associated with the part, or enter a specific language. See Table 2 on page 45 for a list of part types and their associated languages.

The PARTL parameter is where you specify if you want to add the changed part to the part-list part. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the part whose part information is being changed will not be added to a part-list part.

A part-list part must be specified if:

- The group specified on the command was created with PARTLREQ(*YES)
- The type specified is not PARTL

If a part-list part is specified on the CHGPARTINF command, then:

- The part-list part must exist in the default search path starting from the group in which the part information is being changed.
- The name of the part whose part information is being changed is added to the part-list part, if it does not already exist. This will be updated even if

LANG(*SAME), ACCKEY(*SAME), and TEXT(*SAME) are specified. (This may happen even if the command failed to change the part information.)

If the type specified is PARTL, then the PARTL parameter is ignored. In the other cases, the PARTL parameter results in the specified part-list part being updated.

To change the text description of a part, use the TEXT parameter. Use TEXT(*SAME) to keep the same description; TEXT(*BLANK) to change the current description to blank; or TEXT(*description*) to enter a new description.

The text description of an Application Development Manager/400 part can be up to 80 characters long, whereas the text description of its associated native object can only be a maximum of 50 characters long. The object's text description is truncated to a maximum of 50 characters, when a part text description is changed using the CHGPARTINF command.

A part's access key can be changed with the ACCKEY parameter. This allows a new user profile to be identified as the user profile that holds a part, or the access key lock can be removed, making the part available to other users. However, only a project administrator can change the ACCKEY parameter.

Example: The following command changes the language of the part BIWKLY to LF and its text description to blank.

Using the CHGPARTINF command

```
CHGPARTINF PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(DDSSRC) PART(BIWKLY)
           LANG(LF) ACCKEY(*SAME) TEXT(*BLANK)
```

Using the Programming Development Manager utility

Select option 13 (Change information) from the Work with Parts Using PDM display.

Printing Part Information (PRTPARTINF)

The Print Part Information (PRTPARTINF) command prints the following information about a part's characteristics:

- Project, group, type, and part
- Language used to compile the part
- Who changed the part last
- Date and time the contents of the part was last changed
- Date and time the part was created
- Promote code assigned to the part
- Drawdown lock
- Access key
- Corresponding AS/400 name (library, object, type, member)
- Group that contains the lowest occurrence of the part in that branch of the project hierarchy, having the same promote code as this part

- User profile of the person who has the lowest occurrence of the part in that branch of the project hierarchy checked out, having the same promote code as this part
- Text that describes the part

Required Parameters

Specify the project, group, type, and part name of the part whose information you want to print. See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types that can be specified on the PRTPARTINF command.

Optional Parameters

Use the OUTPUT parameter to indicate where the output should go. The default is OUTPUT(*PRINT). The report is spooled to the print device for this job. OUTPUT(*OUTFILE) directs the output to an output file.

The record format of the output file is the same as that used in the system-supplied database file QALYPARTI in library QADM.

Example: You can direct the output to an output file by specifying a command such as the following.

Using the PRTPARTINF command

```
PRTPARTINF PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
           OUTPUT(*OUTFILE) OUTFILE(*LIBL/FILE1)
           OUTMBR(*FIRST *ADD) SCAN(*YES)
```

Using the Programming Development Manager utility

Select option 26 (Print information) on the Work with Parts Using PDM display.

The library list is used to determine where to store the output file called FILE1. The first member in FILE1 receives the output and adds the output data to the existing records.

The SCAN parameter on the PRTPARTINF command specifies whether to search the project hierarchy for the part if it is not found in the group you specified. The default is SCAN(*YES).

Example: The following command prints the part information for the RPGSRC part called BIWKLY.

```
PRTPARTINF PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
           OUTPUT(*PRINT) SCAN(*YES)
```

The report is spooled to the print device for this job. Figure 30 on page 99 shows the sample report.

```
5716PW1   V3R7M0          Application Development Manager/400 - Part Information Report  11/08/96  12:00:00  Page . . :   0001

Project . . . . . :   PAYROLL
Group . . . . . :   DEVELOPER1
Type  . . . . . :   RPGSRC
Part  . . . . . :   BIWKLY

Language  . . . . . :   RPG           1
Last changed (user id)  . . . . . :   SMITH         2
Date last changed . . . . . :   11/08/96  12:00:00   3
Date created  . . . . . :   11/08/96  12:00:00   4
Promote code  . . . . . :   MASTER        5
Drawdown lock . . . . . :                 6
Access key (holder) . . . . . :           7
System name:
  Object  . . . . . :   QRPGSRC       8
    Library . . . . . :     PAY.DEV1    9
  Type  . . . . . :   *FILE         10
  Member  . . . . . :   BIWKLY        11
Checked out to PWS  . . . . . :   No          12
Group of lowest occurrence
  of part . . . . . :             13
Holder of lowest-occurring
  part  . . . . . :               14
Text  . . . . . :                 15
                      * * * * * E N D  O F  L I S T I N G * * * * *
```

Figure 30. Sample of the Part Information Report

The following describes the information in the part information report:

1 The language of the part.

2 The user ID of the person who last changed the part.

3 The date and time the part was last changed. It is possible that the date and time the part was last changed may be earlier than the date and time the part was created. This occurs when you check a part out from a group that is higher in the project hierarchy. This is done so that the build process does not build the part unnecessarily. The date and time the part was last changed is that of the part you copied from the group that is higher in the project hierarchy.

4 The date and time the part was created.

5 The promote code associated with the part.

6 The drawdown lock for the part; either *NONE or the name of the group.

7 The access key, or user profile, of the person holding the part. This is the user profile of the person who has the part checked out. When a part is checked out by someone, no one else is able to update it.

8 The corresponding AS/400 system object name.

9 The corresponding AS/400 system library name.

10 The corresponding AS/400 system type.

11 The corresponding AS/400 system member name. This line does not appear on the report if the part is not a source member.

12 Indicates whether the part is checked out to your programmable work station.

13 Group that contains the lowest occurrence of the part in that branch of the project hierarchy having the same promote code as this part.

14 User profile of the person who has the lowest occurrence of the part in that branch of the project hierarchy checked out, having the same promote code as this part.

15 Text that describes the part.

Retrieving Part Information (RTVPARTINF)

The Retrieve Part Information (RTVPARTINF) command retrieves a fully qualified AS/400 object or member name for a given Application Development Manager/400 four-part name, within a CL or REXX program. If the part is an object, this command returns the library name, object name, and object type. If the part is a member, this command returns the library, file, and member names. This command can only be run as part of a CL or REXX program.

On the RTVPARTINF command, specify the name of the project, group, type, and part, as well as the CL variables that are to receive the library name, object name, object type, and member name. (See Appendix B, “Part Types and Their Relationship to Commands” for a list of part types for which part information can be retrieved using the RTVPARTINF command.) The CL variable must begin with an ampersand (&), and can be a maximum of 10 characters.

You can use a CL or REXX program that retrieves the AS/400 object or member name, and use your own tools to perform AS/400 functions on a part. For example, if the Application Development ToolSet/400 product is not installed on your system, you can write a CL or REXX program that retrieves the name of the AS/400 source object or member that corresponds to an Application Development Manager/400 part, and then use this information to call your own editor.

Figure 31 on page 101 shows a sample CL program called EDITPART that illustrates how a source part can be changed when using an editor other than SEU. The sample program checks a part out, retrieves the AS/400 system information about the part, calls the SEU editor to change the part, and checks the part in. The CHKOUTPART command is necessary so that the part remains under Application Development Manager/400 control. You could create a similar program that calls your own tool in the same way that the SEU editor is called in the following program.

This sample program uses Application Development Manager/400 and AS/400 messages. You can do the same in the CL or REXX program that you create. The message log will show the actions the program has performed.

```
/*****************************************************************/
/*                                                               */
/* A sample program that illustrates how a source part could     */
/* be changed when using an editor other than SEU.  The project, */
/* group, type, and part names are passed to the program. The    */
/* program issues the CHKOUTPART command, calls an editor (SEU is*/
/* used as an example), and then issues the CHKINPART command.   */
/*                                                               */
/*****************************************************************/
            PGM        PARM(&PRJ &GRP &TYP &PART)
/*****************************************************************/
/*                                                               */
/* The following variables are the parameters passed to this     */
/* program.  The variables resolve to the name of the project,   */
/* group, type, and part.                                        */
/*                                                               */
/*****************************************************************/
            DCL        VAR(&PRJ) TYPE(*CHAR) LEN(32)
            DCL        VAR(&GRP) TYPE(*CHAR) LEN(32)
            DCL        VAR(&TYP) TYPE(*CHAR) LEN(10)
            DCL        VAR(&PART) TYPE(*CHAR) LEN(10)
/*****************************************************************/
/*                                                               */
/* The following variables receive the information returned from */
/* the RTVPARTINF command.                                       */
/*                                                               */
/*****************************************************************/
            DCL        VAR(&OBJ) TYPE(*CHAR) LEN(10)
            DCL        VAR(&LIB) TYPE(*CHAR) LEN(10)
            DCL        VAR(&TYPE) TYPE(*CHAR) LEN(10)
            DCL        VAR(&MBR) TYPE(*CHAR) LEN(10)
/*****************************************************************/
/*                                                               */
/* The part is checked out to the user running this program.     */
/* Any messages from the Application Development Manager/400     */
/* product are monitored.  The part must be checked out so that  */
/* it remains under the control of the Application Development   */
/* Manager/400 feature.                                          */
/*                                                               */
/*****************************************************************/
            CHKOUTPART PRJ(&PRJ) GRP(&GRP) TYPE(&TYP) PART(&PART)
            MONMSG     MSGID(ADM0000)
/*****************************************************************/
/*                                                               */
/* The RTVPARTINF command retrieves the AS/400 information       */
/* necessary to edit the part.                                   */
/*                                                               */
/*****************************************************************/
            RTVPARTINF PRJ(&PRJ) GRP(&GRP) TYPE(&TYP) PART(&PART) +
                         OBJLIB(&LIB) OBJ(&OBJ) OBJTYPE(&TYPE) +
                         MBR(&MBR)
            MONMSG     MSGID(ADM0000)
/*****************************************************************/
/*                                                               */
/* The editor that is being used can now be called. In this example */
/* it is assumed that the editor can be started from a command.  */
/* In any case, the AS/400 library name is contained in &LIB, the */
/* AS/400 source file name is contained in &OBJ, and the AS/400  */
/* part name is contained in &MBR.  The line below would be      */
/* replaced with the actual call to the editor.                  */
/*                                                               */
/*****************************************************************/
            STRSEU     SRCFILE(&LIB/&OBJ) SRCMBR(&MBR) OPTION(2)
            MONMSG     MSGID(CPF0000)
/*****************************************************************/
/*                                                               */
/* When the editor is ended, the part must be checked in. This   */
/* informs the Application Development Manager/400 feature that  */
/* the part has been changed.                                    */
/*                                                               */
/*****************************************************************/
            CHKINPART  PRJ(&PRJ) GRP(&GRP) TYPE(&TYP) PART(&PART)
            MONMSG     MSGID(ADM0000)
            ENDPGM
```

Figure 31. A Sample CL Program Called EDITPART that Uses the RTVPARTINF Command

Chapter 8. Part-List Parts, Reason Control, and Change Tracking

A **part-list part** is a part with a part type of PARTL that contains a list of parts that can be processed as a logical grouping. You may find this part type useful when you want to:

- Promote a particular set of parts at once
- Build a particular set of parts at once
- Export a particular set of parts at once
- Track a particular set of parts that are changed

This chapter provides information on the following topics:

- Creating a part-list part
- Changing a part-list part
- Displaying a part-list part
- Printing a part-list part
- Keeping track of problems
- Changing the PARTL parameter default

Creating a Part-List Part

You can use the CRTPART command to create a part-list part.

Example: The following command creates a part-list part in the group DEVELOPER1 in the project PAYROLL. The first part created will be PAY000001.

```
CRTPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(PARTL) PART(*GENERATE)
        LANG(*DFT) TEXT('My first part-list part')
```

You must specify a part type of PARTL and a language of *DFT. The default source file parameter does not apply when you create a part of type PARTL.

To facilitate reason control, now you can create part-list parts with their names automatically generated using consistent naming convention. You can use the CRTPART command with PART(*GENERATE) and TYPE(PARTL) to achieve it. See “Creating a New Part (CRTPART)” on page 47 for more details.

Inside the part-list part, you create a list of the parts you want processed as a logical grouping. For example, in the above part-list part you can add the following entries:

```
RPGSRC PROGRAM1
RPGSRC PROGRAM2
RPGSRC PROGRAM3
```

You can also use a generic part type or part name. Rules for the use of an asterisk (*) and question mark (?) are described in “Required Parameters” on page 40. Use *ALL to indicate that all names should be used. Use **ALL to indicate that all names ending in ALL should be used.

The following part-list entry will cause the processing of the same parts that are shown in the previous example.

```
RPGSRC PROGRAM*
```

Keep the following rules in mind when you work with a part-list part:

- You can list any type of part, including other part-list parts, inside a part-list part.
- Parts that are listed more than once are processed only once.
- Parts may or may not be processed in the order in which they appear in the list. The processing depends on the command being processed and the types of parts that are in the list.
- Parts that are listed but do not exist are ignored.
- If a generic type value, such as P*, is specified and the resulting set of parts contains a part of type PARTL, the PRMPART and BLDPART commands expand the part list.
- If PARTL is specified on the TYPE parameter and a generic part value, such as ABC*, is specified on the PART parameter:
  - The PRMPART command expands the part according to the *LIST or *BOTH value specified on the PARTLOPT parameter.
  - The BLDPART command expands the list of parts inside the part of type PARTL.

See the BLDPART, EXPPART, and PRMPART commands for a description of how you use a part-list part with these commands. For all the other part development commands, the processing of parts of type PARTL is the same as for other part types; that is, the PARTL part itself is processed, not the parts listed in it. See Appendix B, “Part Types and Their Relationship to Commands” on page 247 which shows the commands that allow PARTL to be specified on the TYPE parameter.

Changing a Part-List Part

When you want to change a part of type PARTL using the CHGPART command, a system-supplied Data File Utility (DFU) program is called and the following display appears. DFU is an Application Development ToolSet/400 utility that is used with Application Development Manager/400 feature. For more information about this utility, refer to *ADTS/400: Data File Utility*.

The following display appears when you use the CHGPART command to display a part of type PARTL.

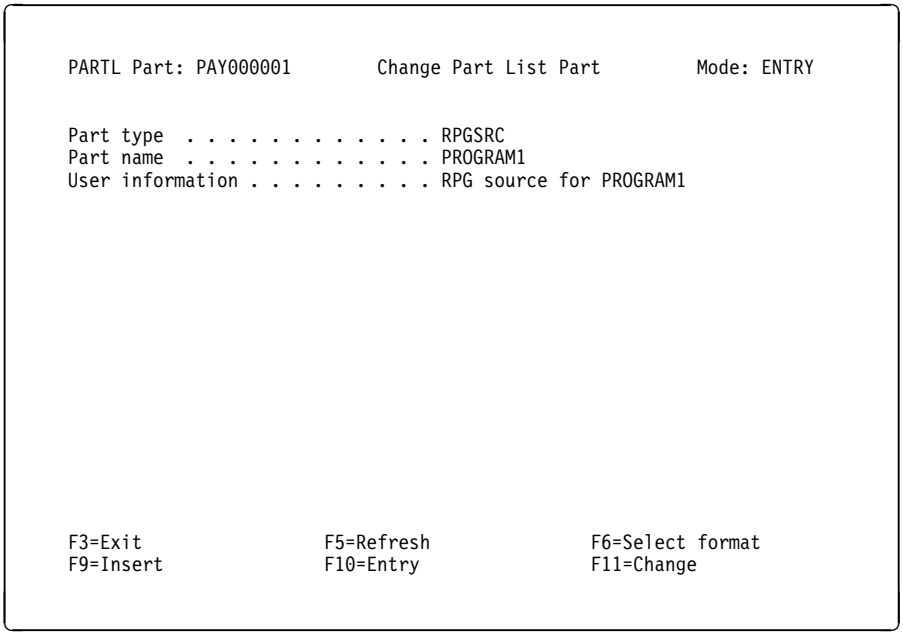

Figure 32. The Change Part List Part Display

Type values for the *Part type* and *Part name* fields. If the values that you specify already exist in the part list, you are automatically put into change mode. This allows you to update any existing part list entries, and is equivalent to pressing F11=Change. Use the roll keys to scroll up and down through the part-list part entries.

Press F1=Help to see information about this display.

Press F3=Exit to exit the current task and return to the display where you began the task.

Press F5=Refresh to clear the input prompts on the display.

Press F6=Select Format to go to the DFU Select Format Key display. For parts of type PARTL there is only one record format.

Press F9=Insert or F10=Entry to add a new part name to the part list. The part does not have to exist to add its name to the part list.

Press F11=Change to change the user information associated with a part in the part list.

Press F14=Record Advance to go to the next part in the list of parts.

Press F21=Status to see the status of this DFU program.

Press F23=Delete to delete the record for the part currently listed from the part list. Confirm the delete operation by press F23 again.

Also add the following entires to the PAY000001 part-list part:

```
RPGSRC PROGRAM2 RPG source for PROGRAM2
RPGSRC PROGRAM3 RPG source for PROGRAM3
```

Although there are no automatic fields defined for the system-supplied DFU program that is running, the following function keys are defined for these displays.

F18 Auto increment from last change
F19 Auto increment from end of file
F20 Auto record advance
F22 Auto duplicate

Displaying a Part-List Part

The following display appears when you use the DSPPART to display a part of type PARTL.

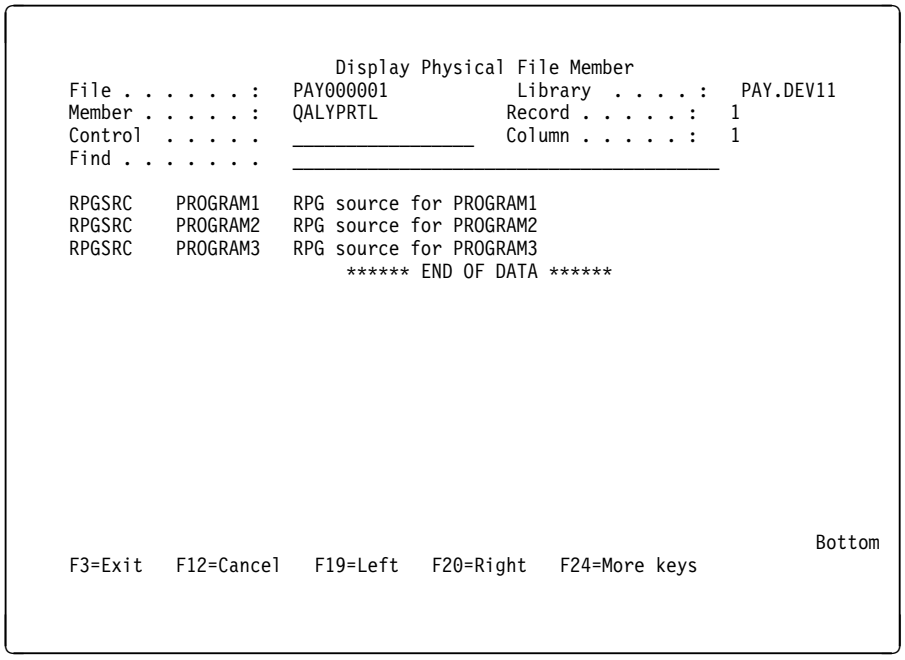

Figure 33. The Display Physical File Member Display

Use this display to see the parts listed in a part-list part of type PARTL.

Printing a Part-List Part

Use the DSPPART command with OUTPUT(*PRINT) to print a part of type PARTL. A spooled file is created with the name QPLYPRTL.

Keeping Track of Problems

An important feature of change management is keeping track of problems. Such problem tracking is also known as **reason control**. The project administrator can turn the reason control on for any specific group, requiring the developer to specify a part-list part name (**reason**) to record the parts being processed in that group. Whenever parts are created, or changed in a group, or whenever parts are promoted into a group, a part-list (reason) is required to be specified. This will help administrators and auditors to more effectively track changes made due to requirements or enhancements requested.

The following Application Development Manager/400 administrator commands provide reason control support:

Change Group (CHGGRP)
Create Group (CRTGRP)
Print Project (PRTPRJ)

On part development commands, specifying a part-list part name on the PARTL parameter causes the parts operated on by the part command to be automatically added to the specified part list, or in the case of the DLTPART command, the parts are automatically removed from the part list, thereby reducing the part-list part maintenance. One exception to this rule is when the part being processed is the same as the part specified on the PARTL parameter. In that case, the name of that part-list part is not added to or removed from the part of type PARTL with the same name (specified on the PARTL parameter).

If PARTLREQ(*YES) is specified when creating or changing a group, developers issuing part commands in that group must specify a valid part-list part name on the command. This gives the administrators the ability to ensure all changes are being tracked to a given requirement or enhancement (reason). If the command fails, you might have to delete the part-list part from the list using the CHGPART command.

The part-list part specified in the PARTL parameter, must exist in the default search path of the group specified on the command. The exception is for the CPYPART command, where it must exist in the default search path of the group that is the target of the copy, as a new part is being created in the target group.

The following Application Development Manager/400 part development commands provide reason control support using the PARTL parameter:

Build Part (BLDPART)
Change Part (CHGPART)
Change Part Information (CHGPARTINF)
Convert Part (CVTPART)
Copy Part (CPYPART)
Create Part (CRTPART)
Delete Part (DLTPART)
Import Part (IMPPART)
Merge Part (MRGPART)
Promote Part (PRMPART)

More than one developer can update the same part-list part at the same time by specifying it in the PARTL parameter on the part development commands.

Changing the PARTL Parameter Default

The system-supplied PARTL parameter default is *NONE for the following Application Development Manager/400 part development commands:

Build Part (BLDPART)
Change Part (CHGPART)
Change Part Information (CHGPARTINF)
Convert Part (CVTPART)
Copy Part (CPYPART)
Create Part (CRTPART)

Delete Part (DLTPART)
Import Part (IMPPART)
Merge Part (MRGPART)
Promote Part (PRMPART)
Rename Part (RNMPART)

Specifying PARTL(*PRV) allows you to use the name of the last part-list part that was used for the specified project, if you are a project administrator. However if you are a developer, specifying PARTL(*PRV) allows you to use the name of the last part-list part that was used for the specified group. PARTL(*PRV) is not valid if you specify it using the QSECOFR user profile.

The PARTL parameter default can be permanently changed from *NONE to *PRV, for any or all of the above-mentioned part development commands in the QSYS library by your system administrator using the CHGCMD command.

Chapter 9. Using Search Paths

This chapter describes:

- Fundamentals of search paths
- What to consider before creating a search-path part
- How to create a search-path part (part of type SCHPTH), including
  - Creating a cross-project search path
  - Creating an external search path to include user libraries
- How search-path parts are processed

Understanding Search Paths

Before you work with commands that make use of search paths, you should have an understanding of them. A typical project has several groups organized within a project hierarchy. A typical project can therefore have several versions of source and output. As defined earlier, a search path is an arrangement of groups that determines the order in which the Application Development Manager/400 feature searches when looking for parts in a project hierarchy. There are four kinds of search paths.

Default search path
The search path up through a branch of the project's group hierarchy, from a particular group through to and including the root group. The **root group** is the first, or top, group in a project hierarchy. In Figure 34 on page 110, the default search path from the group DEVELOPER1 is DEVELOPER1 to TEST to MASTER.

Alternate search path
The search path through a sequence of groups different from that of a branch in the project's hierarchy. In Figure 34 on page 110, an alternate search path could be defined to consist of only the group MASTER, or the groups TEST and MASTER, or perhaps, the DEVELOPER3 and MASTER groups.

Cross-project search path
An alternate search path that includes groups from more than one project. A cross-project search path is useful when you have parts that are referenced by more than one project. You should set up a separate project to hold these common parts, and then include this project in a search-path part for each project that references the common parts.

External search path
An external search path is an alternate search path that includes one or more **external libraries**. External libraries are user libraries found outside the Application Development Manager/400 environment. An external search path is useful when you want to keep a portion of an application outside of Application Development Manager/400 control and still be able to reference it.

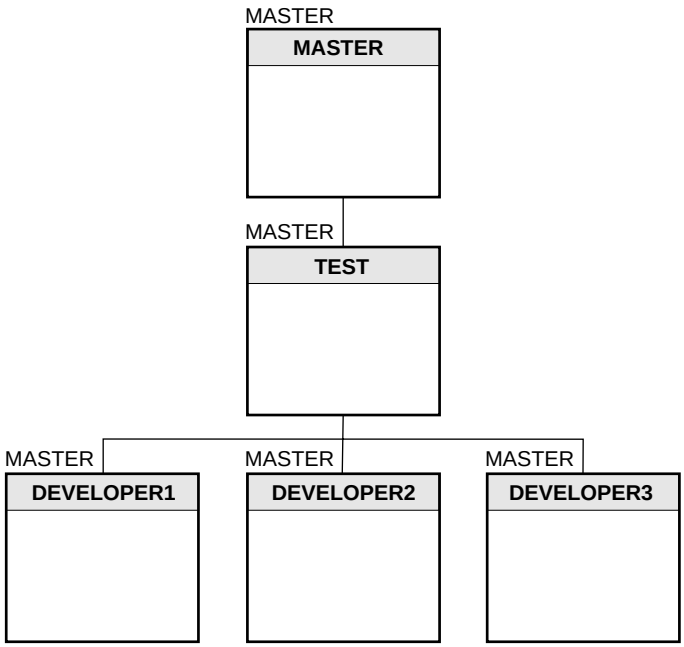

Figure 34. Sample Project Hierarchy Illustrating the Concept of Search Paths

While there is only one project hierarchy for a project, a project can have more than one search path and, accordingly, more than one search-path part (part of type SCHPTH). The SCHPTH part contains the list of groups that determines the order in which groups will be searched for a specified part.

A search-path part is required when you program the following commands to use an alternate or cross-project search path to find parts:

Add Project Library List (ADDPRJLIBL)
Build Part (BLDPART)
Export Part (EXPPART)
Query Part (QRYPART)

A search-path part is also required when you program the following commands to use an external search path to find parts outside the Application Development Manager/400 environment:

Add Project Library List (ADDPRJLIBL)
Build Part (BLDPART)

The ADDPRJLIBL and BLDPART commands change the user portion of the AS/400 library list of the current job to correspond to the order of groups in the default search path or the order of groups listed in a search-path part. Any project libraries already in the user portion of the library list are removed. The project libraries that make up the search path are added to the library list ahead of your AS/400 libraries. The lowest group in the project hierarchy represents the first group that is searched, or, using the library list analogy, the first library in your library list. The BLDPART command restores the library list on completion of the command, the ADDPRJLIBL command does not.

The QRYPART and EXPPART commands do not change the library list, but still use the search path to find parts. Any groups from other projects found in the SCHPTH part are ignored for these commands. SCHPTH parts are useful when you have defined your project hierarchy with multiple branches to manage versions of an application by language, or customer (based on customization). They allow commands such as BLDPART to search for versions of parts in groups not necessarily in the default search path.

Groups and Their Relationship to the AS/400 Library List

The search path within a project hierarchy corresponds to the AS/400 library list. The AS/400 compilers use the library list to find include files and to resolve external references. Figure 35 illustrates the relationship between the project hierarchy and the library list.

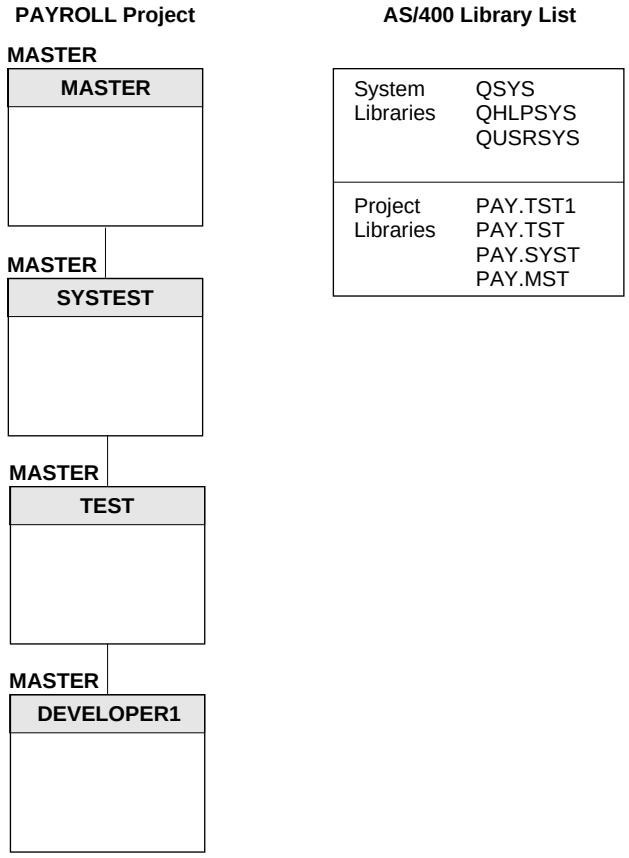

Figure 35. The Payroll Project and its Corresponding AS/400 Library List

There is one AS/400 library for each group. The name of this library consists of the short project name and the short group name separated by a period, for example, PAY.SYST.

Creating a Search-Path Part

A search-path part is required if you need to change the order in which groups are searched for parts; that is, if you need to use an alternate search path, a cross-project search path, or an external search path. The search-path part allows you to create a temporary view of the project hierarchy that is different from the one created by the structure of the project hierarchy itself.

Usually, you do not need to create search-path parts. However, one type of search path you may need is, for example, a cross-project search path. See “Creating a Cross-Project Search Path” on page 113 for more information on this type of search path.

A part of type SCHPTH is created and managed like other Application Development Manager/400 parts, except that you cannot build it. To create a search-path part, use the CRTPART command and specify SCHPTH on the TYPE parameter. The search-path part initially contains the list of groups that make up the default search path, beginning with the group specified on the CRTPART command. Consider Figure 36:

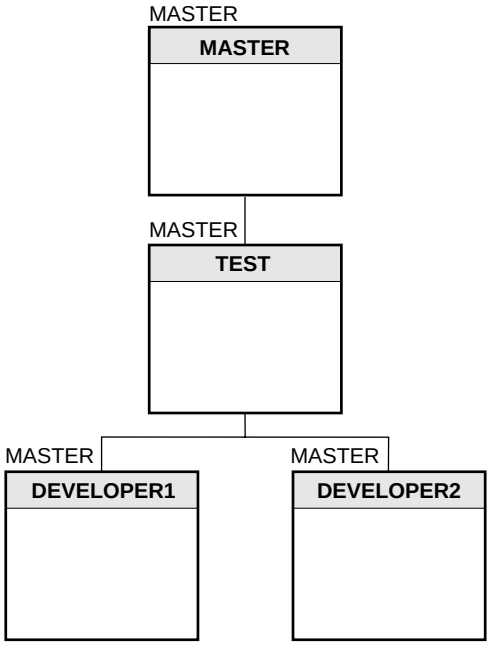

Figure 36. The Default Search Path

For this project hierarchy, the default search path from the group DEVELOPER1 is: DEVELOPER1 to TEST to MASTER. This means that if you are working in the group DEVELOPER1 and you create a new search-path part, it will look like this.

```
PAYROLL DEVELOPER1
PAYROLL TEST
PAYROLL MASTER
```

Developers should not give the name QDFT to search-path parts they create. The project administrator should create the search-path part called QDFT. Its purpose is to allow the use of the value *DFT on commands that have a search path (SCHPTH) parameter.

Each line consists of a project name followed by a group name. The list in the search-path part is searched from top to bottom, similar to the way in which the AS/400 library list is used to find objects. For example, if you specify the group DEVELOPER1 on a command that uses the SCHPTH part shown above, the part is searched for first in the group DEVELOPER1, then TEST, and then MASTER. If you specify the group TEST on a command, then the search for the part would start at TEST and, if not found there, move to MASTER. The search-path part for an alternate search path based on Figure 36 could contain the following information.

```
PAYROLL DEVELOPER2
PAYROLL MASTER
```

When you edit a part of type SCHPTH, keep the following rules in mind:

- The project name appears first. Leading blanks are ignored.
- One or more blanks must follow the project name.
- The group name appears next.
- Any remaining data on the line is ignored.
- No comment lines are allowed.

Remember when creating parts of type SCHPTH that they can be promoted like other parts. This means that many different versions of one search path part can exist in the project hierarchy. Unknown alternate or cross-project search paths could be used accidentally when building parts.

There are certain considerations to remember when using search paths in the Programming Development Manager utility. Refer to “Changing Session Defaults” on page 233 for more information on specifying search paths when using this utility.

Creating a Cross-Project Search Path

A cross-project search path is another kind of alternate search path. Such a search path is useful if you have set up a project to contain files and other parts that are common to several projects. Using a cross-project search path you can quickly find those parts that are common to a particular application. Within a part of type SCHPTH you can define a search path that includes groups from another project. In Figure 37 on page 114, two project hierarchies are used when searching for parts.

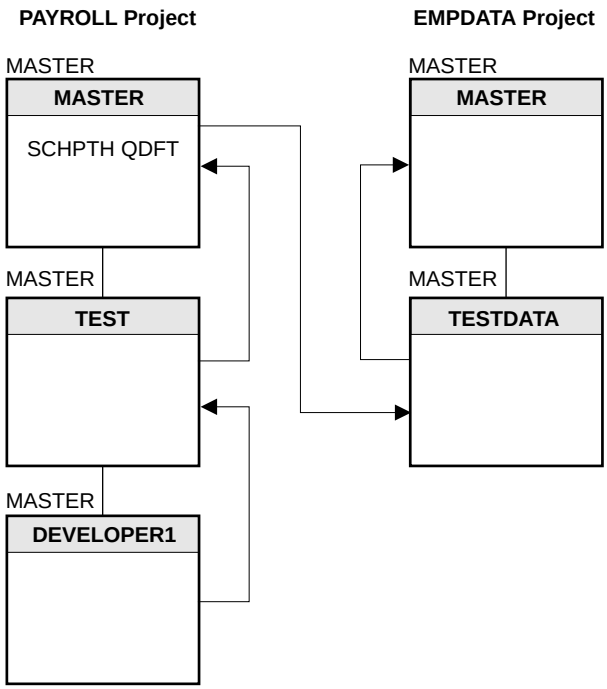

Figure 37. A Cross-Project Search Path

The search-path part PAYROLL MASTER SCHPTH QDFT now contains the following search path information.

```
PAYROLL DEVELOPER1
PAYROLL TEST
PAYROLL MASTER
EMPDATA TESTDATA
EMPDATA MASTER
```

Cross-project search paths are useful if you want to share *common information* across several projects. Common information could include high-level-language programs that are used in more than one application, standard include files that your organization has created and periodically changes, data description specifications (DDS) that are common to more than one application, or field reference files.

A cross-project search path is created and processed in the same way as an alternate search path. However, there are some rules to remember when using a cross-project search path. These are:

- All relevant parts found in the cross-project groups are added to the search path when the BLDPART command is processed.
- The build process searches for and uses parts that represent include files and externally described files in the cross-project groups.
- The ADDPRJLIB and BLDPART commands that use cross-project search paths do not use or search for BLDOPT or SCHPTH parts in the shared project.

- The BLDPART command does not process the parts found in the cross-project groups. For example, if a DDSSRC source part has changed in the shared project, a new FILE part is not created. An existing copy of the FILE part is used by the build process to determine if parts that reference the FILE must be compiled.
- The build process does not issue a warning or error message if a part is stale in the shared project. The administrator should ensure that whenever changes are made in the shared project, the changed parts are built using the BLDPART command.

Creating an External Search Path

An external search path is another kind of alternate search path. Such a search path is useful when you want to access some parts of an application that are outside the Application Development Manager/400 environment. For example, take your vendor's software application, containing a few hundred parts, that is found in a user library. You may want to make changes to some of these parts, but do not want to bring other parts under Application Development Manager/400's control. In this case you can use the external search path to reference the unchanged parts in your user library as Application Development Manager/400 builds this application.

Specifying a special value of *USRLIB as the project name allows you to add one or more external libraries to the search-path part. If *USRLIB is coded, then the library specified on that entry is added to the user portion of the library list that appears immediately after the Application Development Manager/400 project libraries. This algorithm ensures that the build searches for the objects in the Application Development Manager/400 projects first. Only if the objects are not found in these projects, are the user libraries searched.

For example, you may want to add an external library under Application Development Manager/400 control to a project hierarchy similar to the one shown in Figure 36 on page 112. Let us say that this external library is the software library called SOFTLIB. If you want SOFTLIB added to the user portion of the library list, you change the external search-path part to contain the following information:

```
PAYROLL DEVELOPER1
PAYROLL TEST
PAYROLL MASTER
*USRLIB SOFTLIB
```

An external search path is created and processed in the same way as an alternate search path would be. However, there are some rules to remember when using an external search path. These are:

- The search path part is searched for the current group first. All the groups including the user libraries appearing after the current group are added to the library list. It is recommended that all the *USRLIB entries should be made after all the entries for the Application Development Manager/400 project libraries.
- The external libraries are added to the search path when the ADDPRJLIBL command is issued with the search-path part name. The ADDPRJLIBL will ignore an *USRLIB entry, if the user library specified on it already exists in your library list.
- The RMVPRJLIBL cannot remove user libraries added by the ADDPRJLIBL command using the external search path.

- The build process first searches for and uses parts in the Application Development Manager/400 project. Only if the parts are not found there will the build search for the parts in the user library. Remember that the build process will build the source part only if it is found in the Application Development Manager/400 project specified on the BLDPART command.

For more information on using external libraries, see “Adding External Libraries to the Library List” on page 182.

How Search-Path Parts Are Processed

Search paths are processed with the SCAN and SCHPTH parameters. There are three commands that use both the SCAN and SCHPTH parameters (the BLDPART command uses only the SCHPTH parameter):

Add Project Library List (ADDPRJLIBL)
Export Part (EXPPART)
Query Part (QRYPART).

The SCAN and SCHPTH parameters work with each other. Use the SCAN parameter to specify whether to search the project hierarchy for a part if it is not found in the group you specified. SCAN(*YES) searches for a search-path part and uses the search path defined in that part. If it is not found, the default search path is used. SCAN(*NO) limits the search for parts to the group you specify. Use the SCHPTH parameter to indicate the name of a search path part.

If you specify SCAN(*YES), the search path to be used to find the part specified is determined as follows:

1. If the default SCHPTH(*DFT) is specified, this command looks for the SCHPTH QDFT part in the default search path, beginning at the given group.
   - If the SCHPTH QDFT part is found, the search path defined in SCHPTH QDFT is used to search for the part.
   - If SCHPTH QDFT does not exist in the default search path, the default search path itself is used as the search path. All groups in the path from the specified group to the root group are searched until the part is found.
2. When the name of a SCHPTH part is specified, that part is searched for in the default search path, beginning at the given group.
   - If it is found, the search path defined in it is used to search for the part. The SCHPTH part must exist.
   - If the part SCHPTH name is not found, you receive an error message.

Specifying Groups in a Search-Path Part without Listing Them

To indicate you want to use the default search path of a project hierarchy when you are creating a search-path part, use the special value *DFT instead of a group name. It indicates that you want to use the default search path for a project hierarchy. See “Changing Session Defaults” on page 233 for a discussion of search paths and the affect they have when you are using the Programming Development Manager utility.

Example: The search path PAYROLL *DFT results in the search path described in Figure 36 on page 112, assuming DEVELOPER1 was specified on the command that you used.

```
PAYROLL DEVELOPER1
PAYROLL TEST
PAYROLL MASTER
```

The *DFT value is a simple way to include all groups in the default search path for a project hierarchy without having to list each group. The *DFT value is resolved as follows:

1. The search path listed inside the search-path part is scanned for the special value *DFT.
2. For each *DFT found, the line containing *DFT is replaced with the names of the project and group pairs in the following manner:
   a. Starting at the *DFT line, and then checking previous lines listed in the search-path part, each line is checked for the name of the project associated with *DFT.
   b. If the same project name is found, the default search path from that project and group pair is used to resolve the value *DFT.
   c. If the same project name is **not** found, the command you are running is checked to see if the project name matches.
   d. If the same project name is specified on the command you are running, the group currently associated with that project is used as the starting point of the default search path that is to replace the value *DFT. If no matching project name is found, an error message is issued.
3. Once all the *DFT values listed in the search path part have been resolved, the search path starts with the project/group pair specified on the command, and then all project/group pairs that follow, as listed in the search-path part. If the project/group pair specified on the command is not in the resolved list, a warning is issued.

Example: Based on Figure 37 on page 114, suppose the following search path was created and that the group TEST was specified on the QRYPART command.

```
PAYROLL DEVELOPER1
PAYROLL *DFT
EMPDATA TESTDATA
EMPDATA *DFT
```

Based on the rules listed above:

1. The value *DFT is found in the search-path part.
2. The first *DFT value is replaced with only the group MASTER from the PAYROLL project.

```
PAYROLL MASTER
```

3. The second *DFT value is replaced with the default search path starting from the group TESTDATA in the EMPDATA project.

```
EMPDATA TESTDATA
EMPDATA MASTER
```

The result is that the QRYPART command searches for parts in the following search path.

```
PAYROLL TEST
PAYROLL MASTER
EMPDATA TESTDATA
EMPDATA MASTER
```

Chapter 10. Working with User-Defined Types

This chapter describes:

- The concept of user-defined types and who uses them
- Commands you use to work with user-defined types (ADDADMTYPE, RMVADMTYPE, PRTADMTYPE)
- Commands you use to work with a user-defined language (ADDADMLANG, RMVADMLANG, CHGADMLANG, PRTADMLANG)
- The actions associated with the user-defined type (CHGADMACN)

Understanding User-Defined Types

You can define your own part types, which may help make this feature more useful for you. These are called user-defined types. A **user-defined type** is a part type that you define so that it is recognized by the Application Development Manager/400 feature. Although the Application Development Manager/400 feature supports the most common object types, member types, and compilers, there may be applications which contain object types that are not supported. You will find this function useful when you want to manage multiple versions of parts, of if you want to do the following:

- Add a user-defined source member type for a compiler language not currently supported, such as PL/1 or FORTRAN
- Add a user-defined object type for object types not currently supported
- Add a user-defined type that is not to be stored in an AS/400 object or source file member in an Application Development Manager/400 project, to allow CASE tools that need to manage a version of their database model to interact with the Application Development Manager/400 feature

Adding a User-Defined Source Member Type (ADDADMTYPE)

Use the Add ADM Type (ADDADMTYPE) command to add a source member type. To do this, specify *MBR on the SYSTYPE parameter. You might want to add a source member part type if you are using a system-supplied or user-defined compiler that is not supported by the Application Development Manager/400 feature. You can achieve this as follows:

- To be able to create and build parts of the user-defined type with the CRTPART and BLDPART commands, you use the Add ADM Language (ADDADMLANG) command to add a language to a user-defined type, and to define the command that the BLDPART command is to use to compile parts of the user-defined type and language. If you want to compile parts of the user-defined type, the compiler must write to the Application Development Manager/400 APIs described in the *System API Reference*.
- If your compiler does *not* write to the Application Development Manager/400 APIs, you are only able to maintain versions of your source. You will not be able to build parts of the user-defined type.

You can, however, even if you still define the type and language for your user-defined type, have a compiler that does not write to the Application Development Manager/400 APIs. For example, you can add the user-defined type ZPASSRC and define the language PASCAL for the type.

Another reason you might need to add a user-defined type is to accommodate preprocessing. For example, you may need to add a user-defined type called RPT, so that you can store Auto Report Program source in Application Development Manager/400 projects. When you define a language for this type, you must specify the Create Auto Report Program (CRTRPTPGM) command for the BLDCMD parameter on the Add ADM Language (ADDADMLANG) command. Because the CRTRPTPGM command calls CRTRPGPGM, which is enabled for the Application Development Manager/400 feature, you can use BLDPART to compile your RPT parts. Refer to “Adding the User-Defined Type RPT—An Example” on page 135 for more information.

If you have an RPG/38 compiler, for example, you might want to add a System/38 user-defined source member type. You can then create parts of the user-defined type with the CRTPART command. You are not able to define a build command on the BLDCMD parameter of the ADDADMLANG command. The RPG/38 compiler is not enabled to the Application Development Manager/400 feature. You can maintain different versions of your source parts, but to compile parts of the user-defined type you must do one of the following tasks:

- Use the ADDPRJLIBL command to add the group in which the parts reside to the AS/400 library list, and then call the compiler specifying the library in which you want to store the output *PGM object.
- Specify *YES on the SCAN parameter of the EXPPART command to export the parts to be compiled to an AS/400 library, and then call the compiler specifying the name of the library to which you exported the parts.

Adding a User-Defined Object Type

Use the Add ADM Type (ADDADMTYPE) command to add a user-defined object type; for example, part type ZCSI with object type *CSI. Specify a name on the SYSTYPE parameter to add a user-defined object type. Any AS/400 object type that is not restricted to a particular library, for example QSYS or #LIBRARY, and is not a *FILE can have a user-defined type defined for it.

Use the CHGADMACN command in conjunction with the ADDADMTYPE command to define the actions that the Application Development Manager/400 feature must use to work with parts of the user-defined type.

Adding a User-Defined Type Not Stored in an Object or Source File Member

Use the Add ADM Type (ADDADMTYPE) command to add a user-defined type that is not to be stored in an AS/400 object or source file member in an Application Development Manager/400 project. This feature keeps track of each part, but the entity itself is stored outside the control of this feature.

Specify *NONE on the SYSTYPE parameter of the ADDADMTYPE command to indicate that a part of the user-defined type is not to be stored in an AS/400 object or source file member in an Application Development Manager/400 project.

One case in which you might want to add such a user-defined type is if you work with a code-generating tool. You can create a user-defined type in which to store your source code. When you compile the source, it might create one or more members. The members will correspond to either system-supplied or user-defined part types.

Adding a User-Defined Type—The Basic Steps

There are three steps to follow to add a user-defined type:

1. Use the ADDADMTYPE command to add a user-defined type.
2. Define a language using the ADDADMLANG command, to be able to create and build parts with the user-defined type.
3. Define the actions that the Application Development Manager/400 feature needs to use to manipulate parts that have the user-defined type. Use the CHGADMACN command for this. In some cases, there may be no need to use the CHGADMACN command. The ADDADMTYPE command prefills actions when the type is stored in objects for which there are already part types; for example, members and data areas.

Note

When defining user-defined part types, you must ensure that the actions you define will operate against the object, member, or entity that corresponds to the specified part. If you fail to do this, unpredictable results may occur.

Proper use of substitution variables will ensure that the corresponding object, member, or entity is operated against. See Appendix D, “Substitution Variables” for a description of the various substitution variables.

The following list shows all the commands you can use to define and work with user-defined types:

ADDADMLANG	Add a language to a user-defined part type
ADDADMTYPE	Add a user-defined type
CHGADMACN	Change action definitions for a user-defined type
CHGADMLANG	Change a user-defined part language
PRTADMLANG	Print user-defined part language information
PRTADMTYPE	Print user-defined type information
RMVADMLANG	Remove a user-defined part language
RMVADMTYPE	Remove a user-define type

All of these commands, except PRTADMTYPE and PRTADMLANG, are logged in the project log against project *ALL. The PRTPRJLOG command is changed to print all log records that either match the requested project or have *ALL for a project value.

Note: All of these commands, except PRTADMTYPE and PRTADMLANG, require *ALLOBJ authority. You specify this on the SPCAUT parameter of the Create User Profile (CRTUSRPRF) command.

Working with a User-Defined Type

The following commands let you work with user-defined types:

ADDADMTYPE	Add a user-defined type
PRTADMTYPE	Print user-defined type information
RMVADMTYPE	Remove a user-defined type

Adding a User-Defined Type (ADDADMTYPE)

Use the Add ADM Type (ADDADMTYPE) command to add a user-defined type to the Application Development Manager/400 feature. Once added, the user-defined type can be used by all developers in all projects.

You must have *ALLOBJ authority to use this command. This value is specified on the SPCAUT parameter of the Create User Profile (CRTUSRPRF) command.

When a user-defined type is added, no language is assigned to that type. You must define a language using the ADDADMLANG command. See "Adding a Language to a User-Defined Type (ADDADMLANG)" on page 125 for information on this command.

Required Parameters

Specify the user-defined type and the manner in which parts of the user-defined type are to be stored on the AS/400 system. It is recommended that your user-defined types begin with the letter Z. This ensures your user-defined types do not have the same name as any future system-supplied part types. If a future system-supplied type has the same name as a user-defined type, the system-supplied type will take precedence. See Appendix C, "Naming Rules" for the naming rules for a part type.

The object type to store a part of the user-defined type can be the name of an AS/400 object, *MBR, or you can specify that parts of the user-defined type are not to be stored in an AS/400 object or source file member in an Application Development Manager/400 project, or the object type can be an AS/400 source file member.

If you specify a name on the SYSTYPE parameter, a part of the user-defined type is stored in an AS/400 object type. All AS/400 object types that are not restricted to a particular library, are allowed. For example, objects which must exist in the libraries QSYS or #LIBRARY are not allowed. Also, objects of type *FILE are not allowed.

If you choose the value *NONE on the SYSTYPE parameter, a part of the user-defined type is not stored in an AS/400 object or source file member within an Application Development Manager/400 project. This feature keeps track of each part, but the entity itself is stored outside this feature. Parts of the user-defined type are manipulated by the actions you define with the CHGADMACN command.

If you choose the value *MBR, a part of the user-defined type is stored in an AS/400 source file member.

Optional Parameters

You can specify these optional parameters:

- Default source physical file where a part of the user-defined type is to be stored. DFTSRCF(QTXTSRC) is the default for user-defined types that are source members. You can specify the name of a default source file.
- Default record length for any source physical file used to store a part with the user-defined type. DFTRCDLEN(92) is the default. You can specify the default record length for the source physical file (13 to 32678 bytes).
- Whether the part type can be included by other parts. The default is ALWINC(*NO); parts of the user-defined type are not included by other parts.
- Whether the part type can include other parts. The default is INCLUDES(*NONE); parts of the user-defined type cannot include other parts. If you specify a name, the user-defined type must be defined with ALWINC(*YES) or it must be a system-supplied type that is an include.

The values you specify for the ALWINC and INCLUDES parameters are used by the BLDPART command.

Removing a User-Defined Type (RMVADMTYPE)

Use the Remove ADM Type (RMVADMTYPE) command to remove a user-defined type. This is only successful if there are no parts of the user-defined type existing in any project and if no other user-defined type has that specific type defined on the BLDOUTTYPE parameter of the ADDADMLANG command.

You must have *ALLOBJ authority to use this command. This value is specified on the SPCAUT parameter of the Create User Profile (CRTUSRPRF) command.

Simply enter the type to be removed on the RMVADMTYPE command.

If you want to obtain lists of the parts that exist with the type that you want to remove, specify *ALL on the WRKPRJPDM command and you are presented with the Work with Projects Using PDM display. It lists all of the projects to which you are authorized. For each project that is listed, use the QRYPART command and enter your user-defined type in the TYPE parameter. This lists any parts that exist for that user-defined type.

Printing User-Defined Type Information (PRTADMTYPE)

Use the Print ADM Type (PRTADMTYPE) command to run a report that lists any or all of the user-defined types defined for the Application Development Manager/400 feature. This report reflects all the parameter values you chose when you added a user-defined type with the ADDADMTYPE command, except for the actions defined for the type. You use the CHGADMACN command to define these. See Figure 38 on page 124 for a sample report.

Required Parameter

Specify the part type whose information you want to print. TYPE(*ALL) prints the information for all user-defined types. You can specify the name of a specific part type.

Optional Parameters

Use the OUTPUT parameter to indicate where the output should go. The default is OUTPUT(*PRINT). The report is spooled to the print device for this job. OUTPUT(*OUTFILE) directs the output to an output file.

The record format of the output file is the same as that used in the system-supplied database file called QALYPTYP in library QADM.

Example: You can direct the output to an output file by specifying a command similar to this one.

```
PRTADMTYPE TYPE(ZPASSRC) OUTPUT(*OUTFILE) OUTFILE(*LIBL/FILE1)
           OUTMBR(*FIRST *ADD)
```

The library list is used to determine where to store the output file called FILE1. The first member in FILE1 receives the output and adds the output data to the existing records.

Example: The following command prints the user-defined type information for the type PASSRC.

```
PRTADMTYPE TYPE(ZPASSRC) OUTPUT(*PRINT)
```

The report is spooled to the print device for this job. Figure 38 shows the sample report.

```
5716PW1   V3R7M0        Application Development Manager/400 - Print ADM Type          11/08/96    15:30:21          Page . . :  0001

Type . . . . . . . . . . . . . :       ZPASSRC      1
System type  . . . . . . . . . :       *MBR         2
Default source file  . . . . . :       QTXTSRC      3
Default record length  . . . . :       00092        4
Allow type to be included  . . :       No           5
Includes part type . . . . . . :       *NONE        6
*CRT command . . . . . . . . . :       QSYS/ADDPFM FILE(&L/&F) MBR(&ZN)      7

*DLT command . . . . . . . . . :       QSYS/RMVM FILE(&L/&F) MBR(&ZN)        8

*MOV command . . . . . . . . . :       *NONE        9

*CPY command . . . . . . . . . :       QSYS/CPYF FROMFILE(&L/&F) TOFILE(&ZI/&ZF) FROMMBR(&ZN) TOMBR(&ZJ) CRTFILE(*YES) MBROPT(*RE
                                       PLACE) FMTOPT(*MAP)                                  10
*CHG command . . . . . . . . . :       QSYS/STRSEU SRCFILE(&L/&F) SRCMBR(&ZN) OPTION(2)     11

*DSP command . . . . . . . . . :       QSYS/STRSEU SRCFILE(&L/&F) SRCMBR(&ZN) OPTION(5)     12

*PRT command . . . . . . . . . :       QSYS/STRSEU SRCFILE(&L/&F) SRCMBR(&ZN) OPTION(6)     13
```

Figure 38. Sample of a Print ADM Type Report

The following list describes the information in the Print ADM Type report:

1 User-defined type that was specified on the TYPE parameter of the ADDADMTYPE command.

2 System type that was specified on the SYSTYPE parameter of the ADDADMTYPE command.

3 Default source file that was specified on the DFTSRCF parameter of the ADDADMTYPE command.

4 Default record length that was specified on the DFTRCDLEN parameter of the ADDADMTYPE command.

5 Allow parts of the user-defined type to be included by other parts with user-defined part types. This is the value that was specified on the ALWINC parameter of the ADDADMTYPE command.

6 Allow parts of the user-defined type to include other parts. This is the value that was specified on the INCLUDES parameter of the ADDADMTYPE command.

7 The value or create command that was specified on the CMD parameter of the CHGADMACN command.

8 The value or delete command that was specified on the CMD parameter of the CHGADMACN command.

9 The value or move command that was specified on the CMD parameter of the CHGADMACN command.

10 The value or copy command that was specified on the CMD parameter of the CHGADMACN command.

11 The value or change command that was specified on the CMD parameter of the CHGADMACN command.

12 The value or display command that was specified on the CMD parameter of the CHGADMACN command.

13 The value or print command that was specified on the CMD parameter of the CHGADMACN command.

Working with a User-Defined Language

The following commands let you work with a user-defined language:

ADDADMLANG	Add a user-defined language
CHGADMLANG	Change a user-defined language
PRTADMLANG	Print language information
RMVADMLANG	Remove a user-defined language

Adding a Language to a User-Defined Type (ADDADMLANG)

Use the Add ADM Language (ADDADMLANG) command to add a user-defined language to the user-defined type. To be able to create and build parts of the user-defined type with the BLDPART command, you use the Add ADM Language (ADDADMLANG) command to add a language and to define the command that the BLDPART command is to use to compile parts of the user-defined type and language.

Once added, the user-defined language can be used by all developers in all projects. Only user-defined types that represent members that are not includes or that have no object or member (that is, *NONE is specified on the SYSTYPE parameter of the ADDADMTYPE command) can have a command defined on the BLDCMD parameter.

You must have *ALLOBJ authority to use this command. This value is specified on the SPCAUT parameter of the Create User Profile (CRTUSRPRF) command.

Required Parameters

Specify the user-defined type on the TYPE parameter. Specify a language or *NONE on the LANG parameter.

If the language being added is for a user-defined type that corresponds to an AS/400 object that has attributes, it is recommended that the language name be the same as the attribute name. And, if the language being added is for a user-defined type that corresponds to a source member, specify the same language as the member type. This ensures that the IMPPART command imports both objects and source correctly when importing sets of objects or members. For example, a user-defined type of ZPGM with a language of PLI corresponds to object type of *PGM with an attribute of PLI.

Optional Parameters

You can specify these optional parameters:

- The command the build process should use to compile parts with the type and language combination you specified.

  The default is BLDCMD(*NONE). This means that the part type and language cannot be built with the BLDPART command.

  In the following cases, *NONE must be specified on the BLDCMD parameter:

  – If the language being added is *NONE

  – If the user-defined type specified on the TYPE parameter is stored in an object (that is, if the name of an AS/400 object type is specified on the SYSTYPE parameter of the ADDADMTYPE command)

  – If the user-defined type is an include (defined with ALWINC(*YES) on the ADDADMTYPE command)

  You can specify the command string to use to build parts of the user-defined type and language, if the user-defined type is a *MBR or SYSTYPE *NONE. The command string can be up to 250 characters long. Comments should not be entered in the command string. The BLDCMD parameter cannot be prompted. If the compile command you define does not replace the existing output parts of the build process, the BLDPART command fails when it tries to build a part of the user-defined type specified on the BLDPART command.

  You can use substitution variables in the command string. They are replaced with actual values before the command is issued. All the substitution variables valid for the CMD parameter of the CHGADMACN command are valid for this parameter as well. Appendix D, “Substitution Variables” on page 259 describes all the substitution variables and indicates which ones are allowed on the BLDCMD parameter.

  In certain instances, the Application Development Manager/400 feature cannot determine the information with which to replace the substitution variable. For example, the &ZH substitution variable cannot be replaced with the name of the part being moved or copied if the action you are defining is a delete action. Here, it would not make sense to use the substitution variable &ZH. The &ZH substitution variable cannot be replaced for the scan value if you are using the BLDCMD parameter. In this case, it does not resolve if the required information is not available.

Example: This example assumes you are using a user-defined RPG compiler called ZRPG, for example, and that the compiler writes to the Application Development Manager/400 APIs described in the *System API Reference*.

The following command shows an example of the CRTZRPGPGM compile command defined on the BLDCMD parameter of the ADDADMLANG command.

```
ADDADMLANG TYPE(ZRPGSRC) LANG(ZRPG) BLDCMD('CRTZRPGPGM PGM(&L/&ZN)
           SRCFILE(&L/&F) SRCMBR(&ZN)') BLDOUTTYPE(PGM)
           BLDOUTLANG(*INPUT) SPLFNAME(*INPUT)
```

- **The part type of the output object that the build process generates**

  This parameter is ignored if a type that is an include, a type that is stored in an AS/400 object, or a user-defined type that has no associated object or member is specified on the TYPE parameter. It is not allowed if *NONE is specified on the LANG and BLDCMD parameters. The default is BLDOUTTYPE(*NONE). This means no output part is created.

  You can specify a name for the *PGM output object that the build process generates when building a part of the type and language specified.

- **The language of the output that the build process generates**

  This parameter is ignored if a type that is an include, a type that is stored in an AS/400 object, or a user-defined type that has no associated object or member is specified on the TYPE parameter. It is not allowed if *NONE is specified on the LANG and BLDCMD parameters.

  The default is BLDOUTLANG(*INPUT); the output part generated by the build process is to have the same language as the part that created it. Specify BLDOUTLANG(*NONE) if the output part is to have no language or specify a name that identifies the language to assign the output part.

- **The name of the spool file of the compiler listing the build process saves**

  If one is generated by the compiler and if the SAVLST parameter is specified on the BLDPART command.

  This parameter is ignored if the user-defined type on the TYPE parameter is not stored in a member, if the language specified on the LANG parameter is *NONE, or if the user-defined type is an include.

  The default is SPLFNAME(*INPUT); the spool file of the compiler listing has the same name as the part being built. You can specify that the spool file of the compiler listing is to have the same name as the output part generated by the build process, SPLFNAME(*OUTPUT); that no listing is to be created by the compile command you define on the BLDCMD parameter, SPLFNAME(*NONE); or you can specify the name that the spool file of the compiler listing will have. (The build process does not check for or save a compiler listing.)

Changing a User-Defined Language (CHGADMLANG)

Use the Change ADM Language (CHGADMLANG) command to change the way in which the BLDPART command processes parts with a given user-defined type and language combination. Only user-defined types that represent members that are not includes or that have no object or member (that is, that have *NONE specified on the SYSTYPE parameter of the ADDADMTYPE command) can have a command defined on the BLDCMD parameter.

When the CHGADMLANG command is used to change an existing user-defined language, any parts that were built with the old language definition become stale. The build process recognizes that the language has changed and builds the part again the next time the BLDPART command is issued for parts of that type.

You must have *ALLOBJ authority to use this command. This value is specified on the SPCAUT parameter of the Create User Profile (CRTUSRPRF) command.

Required Parameter

Specify a user-defined type on the TYPE parameter. Specify the name of a user-defined language or *NONE for the language on the LANG parameter.

Optional Parameters

You can specify these optional parameters:

- Command the build process should use to compile parts with the type and language combination you specified

  This parameter is ignored if the type and language combination of the type specified on the TYPE parameter represents a type that is stored in an AS/400 object.

  In the following cases, *NONE must be specified:

  – If the language being added is *NONE
  – If the user-defined type specified on the TYPE parameter is stored in an object (not defined with SYSTYPE(*NONE) or SYSTYPE(*MBR) on the ADDADMTYPE command)
  – If the user-defined type is an include (defined with ALWINC(*YES) on the ADDADMTYPE command)

  The default is BLDCMD(*SAME). The current build command is used. *SAME is replaced with the build command when the CHGADMLANG command is prompted.

  If you specify BLDCMD(*NONE), the type and language combination cannot be built.

  You can specify the command string to use to build parts of the user-defined type and language. The command string can be up to 250 characters long. Comments should not be entered in the command string.

  You can use substitution variables in the command string. They are replaced with actual values before the command is issued. The BLDCMD parameter cannot be prompted. Appendix D, “Substitution Variables” on page 259 describes all the substitution variables and indicates which are allowed on the BLDCMD parameter.

- The type of output that the build process generates

  BLDOUTTYPE(*SAME) is the default. The part type for the output object (*PGM) is used. *SAME is replaced with the name of the output type when the CHGADMLANG command is prompted.

  You can specify the name of a part type for the *PGM output object the build process generates when building a part of the type and language specified on the TYPE and LANG parameters.

  If you specify *NONE, no output part is created by the build process.

- The language of the output that the build process generates

  BLDOUTLANG(*SAME) is the default. The current language of the output part type is used. *SAME is replaced with the actual language when the CHGADMLANG command is prompted.

  You can specify a name for the build output part.

  If you specify BLDOUTLANG(*INPUT), the output part generated by the build process is to have the same language as the part that created it.

  If you specify BLDOUTLANG(*NONE), the output part generated by the build process has no language.

- The name the build process uses for the spool file of the compiler listing that the build process saves

  This name applies if a compiler listing is generated by the compiler and if the SAVLST parameter is specified on the BLDPART command.

  SPLFNAME(*SAME) is the default. The current name of the spool file of the compiler listing is used. *SAME is replaced with the name of the listing when the CHGADMLANG command is prompted.

  You can specify a name for the spool file of the compiler listing.

  If you specify SPLFNAME(*INPUT) or SPLFNAME(*OUTPUT), the spool file of the compiler listing has the same name as the input or output part respectively.

  If you specify SPLFNAME(*NONE), no spool file compiler listing is created by the compile command defined on the BLDCMD parameter. The build process does not process or check for a compiler listing if *NONE is specified.

Removing a User-Defined Language (RMVADMLANG)

Use the Remove ADM Language (RMVADMLANG) command to remove a user-defined language. This command is only successful if there are no parts of the user-defined type and language combination existing in any project and if no other user-defined type has that specific language defined on the BLDOUTLANG parameter of the ADDADMLANG command.

You must have *ALLOBJ authority to use this command. This value is specified on the SPCAUT parameter of the Create User Profile (CRTUSRPRF) command.

Simply enter the user-defined type and language to be removed.

If you specify *NONE, this means the language *NONE is removed. (A language of *NONE indicates the part cannot be built.)

If you specify the name of a language, and you are removing the only language that is defined for the part type, the language parameter is reset to *NONE.

Printing User-Defined Language Information (PRTADMLANG)

Use the Print ADM Language (PRTADMLANG) command to run a report that lists any or all of the user-defined languages defined for a user-defined type. The report shows the type, language, and build command associated with a particular type and language combination. The build command is specified on the ADDADMLANG command. It also shows the type and language of the part generated by the build process as well as the name of the spool file.

Required Parameter

Specify the user-defined type whose language information you want to print.

Specify the user-defined language that is to be removed from the user-defined type specified on the TYPE parameter. You can specify the name of a specific part language; LANG(*ALL) to print information about all user-defined languages; or you can specify LANG(*NONE) to print information about the user-defined language *NONE.

Optional Parameters

Use the OUTPUT parameter to indicate where the output should go. The default is OUTPUT(*PRINT). The report is spooled to the print device for this job. OUTPUT(*OUTFILE) directs the output to an output file.

The database format of the output file is the same as that used in the system-supplied database file called QALYPLNG in library QADM.

Example: You can direct the output to an output file by specifying a command like the following.

```
PRTADMLANG TYPE(ZPASSRC) LANG(*ALL) OUTPUT(*OUTFILE) OUTFILE(*LIBL/FILE1)
           OUTMBR(*FIRST *ADD) SCAN(*YES)
```

The library list is used to determine where to store the output file called FILE1. The first member in FILE1 receives the output and adds the output data to the existing records.

Example: The following command prints all the languages defined for the part type ZPASSRC.

```
PRTADMLANG TYPE(ZPASSRC) LANG(PAS) OUTPUT(PGM)
```

A report similar to the report in Figure 39 is spooled to the print device for this job.

```
5716PW1   V3R7M0        Application Development Manager/400 - Print ADM Language      11/08/96     9:49:55          Page . . :  0001

Type . . . . . :       ZPASSRC     1

                                                                                      Output      Output      Spool File
Language    Build command                                                             Type        Language    Name
PAS         CRTPASPGM PGM(&L/&ZN) SRCFILE(&L/&F) SRCMBR(&ZN)                          PGM         *NONE       *INPUT

2       3                                                                 4   5   6

             * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 39. Sample of a Print ADM Language Report

The following list describes the information in the Print ADM Language report:

1 User-defined type that was specified on the PRTADMLANG command, or one of the types being printed if *ALL is specified

2 Languages that were defined for the user-defined type on the ADDADMLANG command

3 Build command that was defined for the user-defined type and language combination on the ADDADMLANG command

4 Output type that was defined for the user-defined type and language combination on the PRTADMLANG command

5 Output language that was defined for the user-defined type and language combination on the PRTADMLANG command

6 Spool file name that was defined for the user-defined type and language combination on the ADDADMLANG command

Defining the Actions to Use with a User-Defined Type (CHGADMACN)

After you add a user-defined type to the Application Development Manager/400 feature, you can define the actions that this feature must use to work with parts of the user-defined type.

When you specify a user-defined type with the ADDADMTYPE command, all actions are initialized to *NONE for those part types whose SYSTYPE parameter value is *NONE, or a value that no system-supplied type uses. If you define a part type for which the SYSTYPE parameter value on the ADDADMTYPE command is already used by a system-supplied type, the action values are initialized similarly to those for the system-supplied type.

The Change ADM Action (CHGADMACN) command specifies actions for user-defined types. You must have *ALLOBJ authority to use this command. This value is specified on the SPCAUT parameter of the Create User Profile (CRTUSRPRF) command.

"Adding the User-Defined Type ZPASSRC—An Example" on page 134 provides examples of the CHGADMACN command. Table 10 shows the actions to use with a user-defined type, as well as the associated Application Development Manager/400 commands that perform the actions.

Table 10 (Page 1 of 2). Actions and Their Associated Commands

Value on ACTION parameter	Action	Application Development Manager/400 command
*CRT	Create	CRTPART
*DLT	Delete	DLTPART, EXPPART (when REPLACE(*YES) is specified), PRMPART (when SYSTYPE(*MBR) is specified on the ADDADMTYPE command), and RNMPART for any type.
*MOV	Move a part through a project hierarchy	PRMPART

Note: You should preface the name of the *MOV action with the name of the library that contains the command. This ensures that another copy of the command is not issued under adopted authority. You should ensure the command you define for the move action does not provide someone with the opportunity to enter a command.

*CPY	Copy	CHKOUTPART, CPYPART, EXPPART, IMPPART, PRMPART if SYSTYPE(*MBR) is specified on the ADDADMTYPE command, and RNMPART for any type.
*CHG	Change	CHGPART

Table 10 (Page 2 of 2). Actions and Their Associated Commands

Value on ACTION parameter	Action	Application Development Manager/400 command
*DSP	Display	DSPPART
*PRT	Print	DSPPART

Required Parameters

Specify the user-defined type on the TYPE parameter and an action for it on the ACTION parameter. See Table 10 for the values you can choose on the ACTION parameter and the Application Development Manager/400 command associated with each value.

The *CRT action that is defined is prompted when *YES is specified on the PRMPT of the CRTPART command. The *CHG action that is defined is prompted if the name of a user-defined type that is an object is specified on the SYSTYPE parameter of the ADDADMTYPE.

Input to all keyword parameters is inhibited when a command prompts any one of the seven actions. A question mark (?) is inserted before the command string. A question mark and asterisk (?*) are inserted before each keyword parameter in the command string.

It is not recommended that you define actions with the prompting characters (? and ?*).

Optional Parameters

Use the CMD parameter with the ACTION parameter. On the CMD parameter, specify the command you want the Application Development Manager/400 feature to use to perform the action you specified on the ACTION parameter. The CMD parameter cannot be prompted.

The default is CMD(*SAME). The command for the specified action does not change. The Application Development Manager/400 command is used. *SAME is replaced with the command when the CHGADMACN command is prompted (see Table 10). If you specify CMD(*NONE), this indicates that the action specified on the ACTION parameter cannot be performed on the parts of the user-defined type.

You can specify a command string that is to perform the action specified on the ACTION parameter. You can use substitution variables here which are replaced with values before the command is issued.

Sometimes, the Application Development Manager/400 feature cannot determine the information with which to replace the substitution variable. For example, the &ZJ substitution variable cannot be replaced with the name of the part being moved or copied, if the action you are defining is a delete action. In this case, it would not make sense to use the substitution variable &ZJ. The substitution variable does not resolve if the required information is not available. Appendix D, “Substitution Variables” describes all the substitution variables and indicates which are allowed on the CMD parameter.

If the TEXT parameter of any Application Development Manager/400 command is specified, then the TEXT parameter specified on any user-defined-type action is ignored. For example, you can specify the TEXT parameter on the *CRT action, but the TEXT parameter on the CRTPART command will be used when creating a part of this type.

If a user-defined-type action is unable to find a part that was passed to it for processing, the following message should be returned.

```
ADM9004  Part not found by user-defined-type action.
```

The Application Development Manager feature responds to this message by removing the Part Directory Entry associated with the part being processed. The message should only be returned for user-defined-type actions that are defined with SYSTYPE(*NONE).

Adding the RJE Object Type *CSI—An Example

The following steps show how to add a user-defined type of RJE:

1. Use the ADDADMTYPE command to add a user-defined type.

   To define the RJE object type *CSI as a user-defined type, specify the following.

   ```
   ADDADMTYPE TYPE(ZCSI) SYSTYPE(*CSI)
   ```

2. Use the ADDADMLANG command to specify a language for the user-defined type so that the part can be created and built. Substitution variables can be used in the command string. Appendix D, “Substitution Variables” on page 259 describes all the substitution variables.

   To add a language to the user-defined type CSI, specify the following.

   ```
   ADDADMLANG TYPE(ZCSI) LANG(*NONE)
   ```

   A part of type CSI cannot be built, therefore LANG(*NONE) is specified.

3. Use the CHGADMACN command to define the actions that the Application Development Manager/400 feature needs to use to work with parts that have the user-defined type. Substitution variables can be used in the command string. Appendix D, “Substitution Variables” on page 259 describes all the substitution variables.

   To define the create action for a CSI part type, specify the following.

   ```
   CHGADMACN TYPE(ZCSI) ACTION(*CRT) CMD('QSYS/CRTCSI CSI(&L/&ZN)')
   ```

   To define the delete action for a CSI part type, specify the following.

   ```
   CHGADMACN TYPE(ZCSI) ACTION(*DLT) CMD('QSYS/DLTCSI CSI(&L/&ZN)')
   ```

   To define the move action for a CSI part type, specify the following.

   ```
   CHGADMACN TYPE(ZCSI) ACTION(*MOV) CMD('QSYS/MOVOBJ OBJ(&L/&ZN)
             OBJTYPE(*CSI) TOLIB(&ZI)')
   ```

   To define the copy action for a CSI part type, specify the following.

   ```
   CHGADMACN TYPE(ZCSI) ACTION(*CPY) CMD('QSYS/CRTDUPOBJ OBJ(&ZN)
             FROMLIB(&L) OBJTYPE(*CSI) TOLIB(&ZI) NEWOBJ(&ZJ)')
   ```

   To define the change action for a CSI part type, specify the following.

   ```
   CHGADMACN TYPE(ZCSI) ACTION(*CHG) CMD('QSYS/CHGCSI CSI(&L/&ZN)')
   ```

To define the display action for a CSI part type, specify the following.

```
CHGADMACN TYPE(ZCSI) ACTION(*DSP) CMD('QSYS/DSPCSI CSI(&L/&ZN)')
```

To define the print action for a CSI part type, specify the following.

```
CHGADMACN TYPE(ZCSI) ACTION(*PRT) CMD('QSYS/DSPCSI CSI(&L/&ZN)
          OUTPUT(*PRINT)')
```

Adding the User-Defined Type ZPASSRC—An Example

The following steps show how to add a user-defined type of ZPASSRC. This example assumes that the user has a PASCAL compiler that is enabled to the Application Development Manager/400 feature:

1. Use the ADDADMTYPE command to add a user-defined type.

   To define the type ZPASSRC that is to be stored in an AS/400 source file member, specify the following.

   ```
   ADDADMTYPE TYPE(ZPASSRC) SYSTYPE(*MBR)
   ```

2. Use the ADDADMLANG command to specify a language for the user-defined type so that the part can be built.

   To add a language to a user-defined type, specify the following.

   ```
   ADDADMLANG TYPE(ZPASSRC) LANG(PAS) BLDCMD('CRTPASPGM PGM(&L/&ZN)
              SRCFILE(&L/&F) SRCMBR(&ZN)') BLDOUTTYPE(PGM)
              BLDOUTLANG(*INPUT) SPLFNAME(*INPUT)
   ```

3. Use the CHGADMACN command to define the actions that the Application Development Manager/400 feature needs to use to work with parts of the user-defined type. Substitution variables can be used in the command string you defined on the CMD parameter. Appendix D, “Substitution Variables” on page 259 describes all the substitution variables and indicates which are allowed on the CMD parameter.

   This example uses the member type ZPASSRC. Action commands are prefilled in the CMD parameter of the CHGADMACN command. The examples below show the system-supplied prefilled actions. You might want to change these prefilled actions to suit your own needs.

   To define the create action for a ZPASSRC part type, specify the following.

   ```
   CHGADMACN TYPE(ZPASSRC) ACTION(*CRT) CMD('QSYS/ADDPFM FILE(&L/&F)
             MBR(&ZN)')
   ```

   To define the delete action for a ZPASSRC part type, specify the following.

   ```
   CHGADMACN TYPE(ZPASSRC) ACTION(*DLT) CMD('QSYS/RMVM FILE(&L/&F)
             MBR(&ZN)')
   ```

   To define the move action for a ZPASSRC part type, specify the following.

   ```
   CHGADMACN TYPE(ZPASSRC) ACTION(*MOV) CMD(*NONE)
   ```

   To define the copy action for a ZPASSRC part type, specify the following.

   ```
   CHGADMACN TYPE(ZPASSRC) ACTION(*CPY) CMD('QSYS/CPYF FROMFILE(&L/&F)
             TOFILE(&ZI/&ZF) FROMMBR(&ZN) TOMBR(&ZJ) CRTFILE(*YES)
             MBROPT(*REPLACE) FMTOPT(*MAP)')
   ```

   To define the change action for a ZPASSRC part type, specify the following.

```
CHGADMACN TYPE(ZPASSRC) ACTION(*CHG) CMD('QSYS/STRSEU SRCFILE(&L/&F)
          SRCMBR(&ZN) OPTION(2)')
```

To define the display action for a ZPASSRC part type, specify the following.

```
CHGADMACN TYPE(ZPASSRC) ACTION(*DSP) CMD('QSYS/STRSEU SRCFILE(&L/&F)
          SRCMBR(&ZN) OPTION(5)')
```

To define the print action for a ZPASSRC part type, specify the following.

```
CHGADMACN TYPE(ZPASSRC) ACTION(*PRT) CMD('QSYS/STRSEU SRCFILE(&L/&F)
          SRCMBR(&ZN) OPTION(6)')
```

Adding the User-Defined Type RPT—An Example

The following steps show how to add a user-defined type of RPT:

1. Use the ADDADMTYPE command to add a user-defined type.

   To define a part type for RPT includes, specify the following.

   ```
   ADDADMTYPE TYPE(ZRPTINC) SYSTYPE(*MBR) DFTSRCF(QRPGSRC) ALWINC(*YES)
   ```

2. Use the ADDADMLANG command to specify a language for the user-defined type.

   To define a language for RPT includes, specify the following.

   ```
   ADDADMLANG TYPE(ZRPTINC) LANG(RPG)
   ```

3. Use the ADDADMTYPE command to add a user-defined type.

   To define a part type for RPT source, specify the following.

   ```
   ADDADMTYPE TYPE(ZRPTSRC) SYSTYPE(*MBR) DFTSRCF(QRPGSRC)
              INCLUDES(ZRPTINC)
   ```

4. Use the ADDADMLANG command to specify a language for the user-defined type so that the part can be created and built. Substitution variables can be used in the command string. Appendix D, “Substitution Variables” describes all the substitution variables.

   ```
   ADDADMLANG TYPE(ZRPTSRC) LANG(RPG) BLDCMD('CRTRPTPGM PGM(&L/&ZE)
              SRCFILE(&L/&F) SRCMBR(&ZN)') BLDOUTTYPE(PGM) BLDOUTLANG(RPG)
   ```

Chapter 11. Building an Application

This chapter describes:

- Understanding part relationships
- Compilers and preprocessors called by the BLDPART command
- How the BLDPART command determines when to compile a part
- Building for the first time
- Using build options
- How the build process searches for parts
- Using the BLDPART command
- Special build processing for specific part types
- Building a sample application

Understanding Part Relationships

There are several components that make up an application: source that you can compile, output such as a module, program or file that is generated by compiling source. Each of these components is an Application Development Manager/400 part. Part relationships are determined and stored in this feature when you use the BLDPART command.

There are two kinds of relationships: a *creates* relationship and a *dependency* relationship. When a source part is compiled to create an output part such as a program, a *creates* relationship is established. There are various kinds of *dependency* relationships. When you compile a source part and it requires another part to exist, this is a dependency. Files, subprograms, and includes are examples of *dependency* parts. Each time you use the BLDPART command, the information about dependent parts used in the compilation of a part is stored with that compiled part as a *dependency* relationship. The BLDPART command uses these relationships to determine which parts to consider for the build process and which parts require compiling.

Parts that have never been built before do not have relationships, so these parts are built when the BLDPART command is called. Their relationships are added with this first build process, from the information returned from the compiler.

When the BLDPART command successfully processes a part, it saves the information about the relationships among the parts that were used when the part was compiled. A part is considered to be built when its build processing has been successfully completed. Build processing activities include compiling the part, if necessary, and saving the information about this part's relationships with other parts.

Table 11 on page 138 shows part relationships in the Application Development Manager/400 feature. The table headings have the following meanings:

Source of relationship
The source is high-level language source or a C/400 program or module.

Relationship
The relationship between the source and the target of the relationship is either a *creates* or *dependency* relationship.

Relationship name
The name of the relationship between the source and the target; one of *creates, binds, includes, subprogram, references, is based on, references record in,* or *references field in*.

Target of relationship
The part related to the source of the relationship.

Table 11 (Page 1 of 2). Relationships in Application Development Manager/400

Source of Relationship	Relationship	Relationship Name	Target of Relationship
BNDSRC	creates	creates	SRVPGM
CBLSRC, CBLLESRC, CLLESRC, CLPSRC, CSRC, RPGLESRC, RPGSRC	creates	creates	PGM
CBLLESRC, CLLESRC, CSRC (of language CLE), RPGLESRC	creates	creates	MODULE
CBLLESRC of language SQLCBLLE, CSRC of language SQLCLE, RPGLESRC of language SQLRPGLE	creates	creates	MODULE, PGM, SRVPGM
CLDSRC	creates	creates	CLD
CMDSRC	creates	creates	CMD
DDSSRC	creates	creates	FILE
MODULE	creates	creates binds	PGM
PNLSRC of language MENU	creates	creates	MENU of language UIM
PNLSRC of language PNLGRP	creates	creates	PNLGRP
BNDDIR	dependency	references	BNDDIR, MODULE, SRVPGM
CBLINC, CBLLEINC, CBLLESRC, CBLSRC, CINC, CLLESRC, CLPSRC, CSRC, DDSSRC, RPGINC, RPGLESRC, RPGLEINC, RPGSRC	dependency	references	FILE
CMDSRC, DDSSRC, PNLSRC, PNLINC	dependency	references	MSGF
MODULE (CRTPGM)	dependency	references	BNDDIR, MODULE, SRVPGM
CBLINC, CBLSRC	dependency	includes	CBLINC
CBLLEINC, CBLLESRC	dependency	includes	CBLLEINC
CINC, CSRC	dependency	includes	CINC
PNLINC, PNLSRC	dependency	includes	PNLINC
RPGINC, RPGSRC	dependency	includes	RPGINC
RPGLEINC, RPGLESRC	dependency	includes	RPGLEINC
DDSSRC of language LF	dependency	is based on	FILE
CBLINC, CBLSRC, CBLLEINC, CBLLESRC, CINC, CLLESRC, CLPSRC, CSRC, DDSSRC, RPGINC, RPGSRC, RPGLESRC, RPGLEINC	dependency	references record in	FILE

Table 11 (Page 2 of 2). Relationships in Application Development Manager/400

Source of Relationship	Relationship	Relationship Name	Target of Relationship
DDSSRC	dependency	references field in	FILE
User-defined type stored in member (*MBR specified on SYSTYPE parameter of ADDADMTYPE command)	creates	creates	System-supplied or user-defined type stored in an object
User-defined type stored in member (*MBR specified on SYSTYPE parameter of ADDADMTYPE command)	creates / includes	creates / includes	System-supplied or user-defined type stored in an object
User-defined type not stored in object or source file member in a project (*NONE specified on SYSTYPE parameter of ADDADMTYPE command)	creates	creates	System-supplied or user-defined type stored in a member or multiple members

Notes:

- CSRC and PGM of language C and SQLC cannot be built.
- System/36 types cannot be built.

Figure 40 illustrates the concepts in Table 11 on page 138. The parts exist in a small project that consists of only one group; therefore a fully qualified name that gives the project, group, type, and part names is not necessary for this example. For simplicity, only the part type and part name are shown. Outside the boxes a **C** represents a *creates* relationship, and a **D** represents a *dependency* relationship.

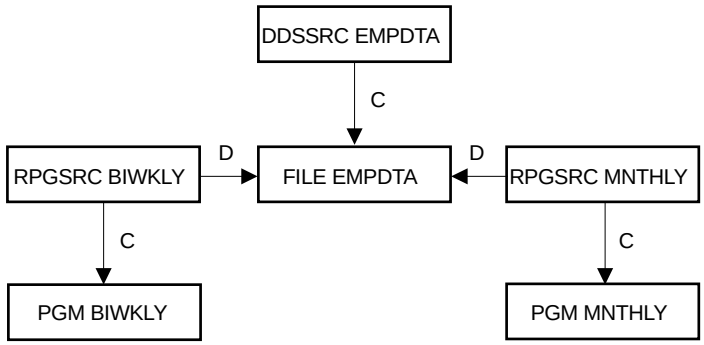

Figure 40. The Parts in a Group and Their Dependencies

The project has the following parts:

DDSSRC EMPDTA-PF — DDS source physical file that is compiled into FILE EMPDTA. DDSSRC EMPDTA *creates* FILE EMPDTA.

FILE EMPDTA-PF — The file created by compiling DDSSRC EMPDTA.

RPGSRC BIWKLY — Source code that is compiled into PGM BIWKLY. RPGSRC BIWKLY has a *dependency* on FILE EMPDTA and *creates* PGM BIWKLY.

PGM BIWKLY	The program created by compiling RPGSRC BIWKLY.
RPGSRC MNTHLY	Source code that is compiled into PGM MNTHLY. RPGSRC MNTHLY has a *dependency* on FILE EMPDTA and *creates* PGM MNTHLY.
PGM MNTHLY	The program created by compiling RPGSRC MNTHLY.

Figure 41 illustrates the same concepts as Figure 40 but uses parts of type CSRC. It shows how a part of type CSRC and language CLE can create a part of type MODULE with the Create C/400 Module (CRTCMOD) command. When the part of type MODULE is bound using the Create Program (CRTPGM) command, it creates a part of type PGM. If you use the Create Bound C Program (CRTBNDC) command with a part of type CSRC, a part of type MODULE is not created, only a part of type PGM is created.

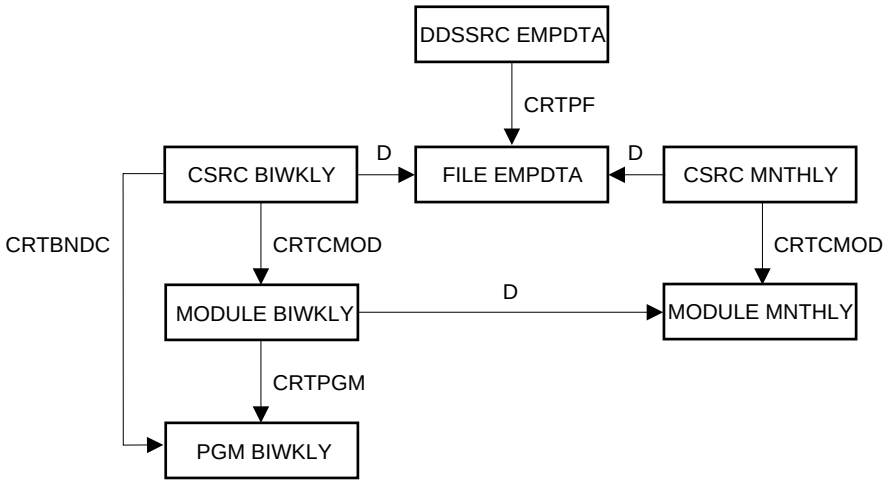

Figure 41. Parts and Their Dependencies in a Group. Parts of type CSRC or type CINC have language CLE.

Compilers and Preprocessors Called by the BLDPART Command

Table 12 on page 141 lists the compilers and preprocessors that can be used with the Application Development Manager/400 feature, and also shows whether a specific compiler can be used when TGTRLS(*PRV) is specified on the compiler or preprocessor command. The BLDPART command can be issued in batch using the SBMJOB command. However, the compilers called from the BLDPART command must be allowed to run in interactive mode. This must be done this way because the BLDPART command processes parts one after the other and needs to process the information returned by the compilers.

Table 12. Compilers and preprocessors that can be used with the Application Development Manager/400 feature

Compiler/ Preprocessor Language	Compiler/Preprocessor OS/400 Command	Supported if *PRV is Specified for Target Release
RPG/400	CRTRPGPGM	Yes
ILE RPG/400	CRTBNDRPG[1], CRTRPGMOD	No
COBOL/400	CRTCBLPGM	Yes
ILE COBOL/400	CRTBNDCBL[1], CRTCBLMOD	No
ILE CL	CRTBNDCL[1], CRTCLMOD	No
ILE C	CRTCMOD[1], CRTBNDC	Yes
DDS[2]	CRTPF, CRTLF, CRTDSPF, CRTPRTF, CRTICFF	Not applicable
CL	CRTCLPGM	Yes
CLD	CRTCLD	Yes
CMD	CRTCMD	Not applicable
DB2 for OS/400 SQL[2]	CRTSQLRPG, CRTSQLCBL, CRTSQLCI	Yes
DB2 for OS/400 SQL[2]	CRTSQLRPGI, CRTSQLCBLI	No
CRTSRVPGM	CRTSRVPGM	Yes
CRTPGM	CRTPGM	Yes
MENU	CRTMNU TYPE(*UIM)	Not applicable
PNLGRP	CRTPNLGRP	Not applicable

Notes:

1. Default command is used by the BLDPART command.
2. Appropriate default compiler command is used based on the part type and the language.

How the BLDPART Command Determines When to Compile a Part

This section describes some of the reasons a part is built when you issue the BLDPART command:

- The source part has not been compiled before. No relationships for the source part exist.
- The BLDOPT part has changed since the part was last built. See “Using Build Options” on page 157 for more information.
- The BLDOPT part that is used for compiling the given part is different from the one used in a previous build.
- A BLDOPT part was used in a previous build, but now the default compile command is being used, or vice versa.
- The FORCE parameter of the BLDPART command is set to *YES. All parts are built whether current or stale.
- The output part is not found in the search path.

- The timestamp for the source part or an include part has changed.

The build report describes the reasons the build process compiles a part. Appendix E, “Build Report Messages” lists the build report *reason for building* messages.

Setting Up the Build Environment

The project administrator sets up the build environment so that as each developer uses the BLDPART command, parts are processed consistently. The project administrator determines the following:

Saving Data: Whether data is saved or deleted when physical data file parts in the application are built is determined by the SAVDTA parameter on the CRTPRJ command. If the project is created with SAVDTA(*YES), the data that is associated with the physical data files is saved when you build these files. If the project is created with SAVDTA(*NO), the associated data is deleted.

The value for the SAVDTA parameter can be changed on the CHGPRJ command if it no longer meets your requirements. The administrator determines whether the data is to be saved. Developers cannot change the SAVDTA parameter.

Establishing BLDPART Guidelines: Guidelines or rules can be established in your organization to encourage developers to build all parts and to verify that parts work as expected before promoting them to the next group in the project hierarchy. In addition, specific BLDPART parameter values can be recommended. For example, developers could be encouraged to use SCOPE(*LIMITED) to build new parts. Once the build operation is successful, the part can be built using SCOPE(*NORMAL).

One way to encourage developers to build parts in a consistent manner is to set up a user-defined option in the Programming Development Manager utility for all developers to use when they are ready to build parts.

Example: The following user-defined option for the BLDPART command indicates a specific search path part and build scope to use when building parts in a development group.

```
BP = "BLDPART PRJ(&ZP) GRP(&ZG) TYPE(&ZT) PART(&ZN)
     SCHPTH(DEVGRP) SCOPE(*LIMITED)"
```

The administrator can establish compiler processing standards by creating a part of type build options (BLDOPT). See “Using Build Options” on page 157 for information on the build options part.

The administrator can create an alternate search path for developers to use. An alternate search path is defined in a part of type SCHPTH. See “Creating a Search-Path Part” on page 112 for more information on search paths.

If your organization uses parts of type SCHPTH and BLDOPT, developers should be discouraged from promoting their own BLDOPT or SCHPTH parts unless they have approval from the administrator or team leader. See “More than One Build Options Part in the Project Hierarchy” on page 166 for information on how these part types are used in the project hierarchy.

Building for the First Time

This section describes building parts for the first time.

Assume that you are working in the group DEVELOPER1. Before you issue the BLDPART command, the parts have no dependency relationships. The following parts in DEVELOPER1 are built using the BLDPART command defaults. *ALL is specified for the PART and TYPE parameters.

In the examples in this chapter, if it is necessary that the part's language be identified, it follows the part name; for example, PGM BIWKLY-RPG.

RPGSRC BIWKLY	Part of a payroll application; an RPG program that references the logical file FILE EMPID-PF.
DDSSRC EMPID-PF	Employee identification; a logical file based on the physical file FILE EMPDTA-PF.
DDSSRC EMPDTA-PF	Employee data; a DDS source physical file.

Figure 42 shows the parts in the group DEVELOPER1 before you issue the BLDPART command.

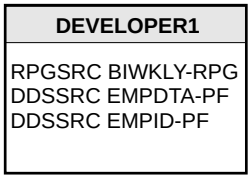

Figure 42. Group DEVELOPER1 Before Using the BLDPART Command

After a successful BLDPART command has been run, the relationships among the parts are established and the source parts have been compiled into the following output parts in DEVELOPER1.

RPGSRC BIWKLY	Compiled into PGM BIWKLY-RPG
DDSSRC EMPID-PF	Compiled into FILE EMPID-PF
DDSSRC EMPDTA-PF	Compiled into FILE EMPDTA-PF

The group DEVELOPER1 now contains the parts listed in Figure 43.

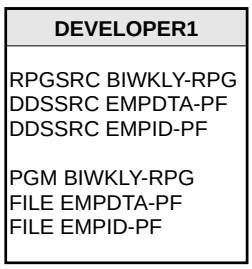

Figure 43. Group DEVELOPER1 After Using the BLDPART Command

The next time you use the BLDPART command, the build process checks the dependency relationships to determine which parts to compile and whether these parts require compiling.

Figure 44 shows the relationships among the parts in the group DEVELOPER1. Again, a **C** represents a *creates* relationship, and a **D** represents a *dependency* relationship.

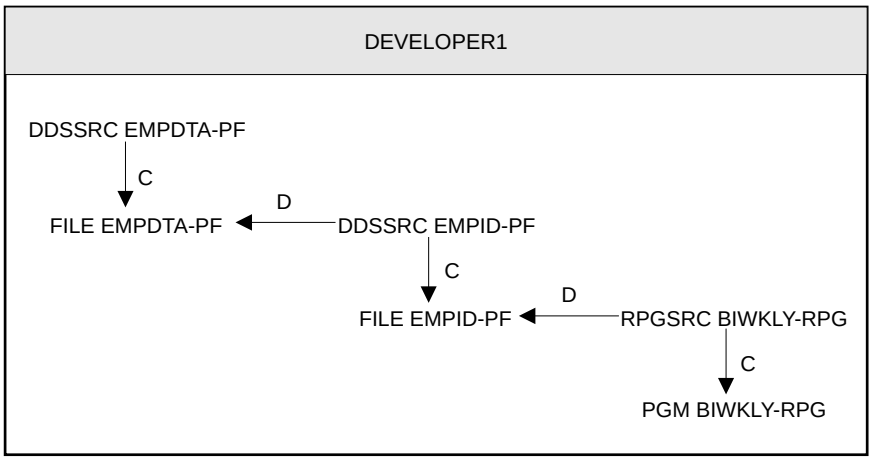

Figure 44. Build Relationships and Result of the Build Process in Group DEVELOPER1

You can build either a source part or an output part on the BLDPART command. The output part is updated, if necessary. You can also specify to build an include part with an *EXTENDED build scope, for example, and all parts that are related to that include part are built.

Note: You can specify file names in your source parts when you are referencing include parts or file parts, but *do not* use specific library names in your source in this case. If you do, the build process may not find the correct version of the part in the project hierarchy.

Use *LIBL instead of the name of a specific library, or the default of *LIBL where required. The build process issues a warning message in the build report if a dependency is used in a compile, and that dependency is not an Application Development Manager/400 part.

After Each Build

After every build, you should:

1. Check the build completion message to see if any parts failed to build, or if any warnings were issued.
2. If one or more parts failed to build, or if there were warnings issued, you should check the build report.
3. Check the job log to determine what may have gone wrong.
4. Check the compiler listing to determine what may have gone wrong.

How the Build Process Searches for Parts

The build process looks through the search path for parts in the same way compilers search the library list on the AS/400 system. In fact, the build process changes the library list to contain the libraries in the search path. When you issue the BLDPART command, the AS/400 library list is changed according to the SCHPTH parameter. After the build process, the AS/400 library list returns to the state it was in before the build process. (For more information on search paths, see Chapter 9, "Using Search Paths" on page 109.) Consider the project hierarchy in Figure 45:

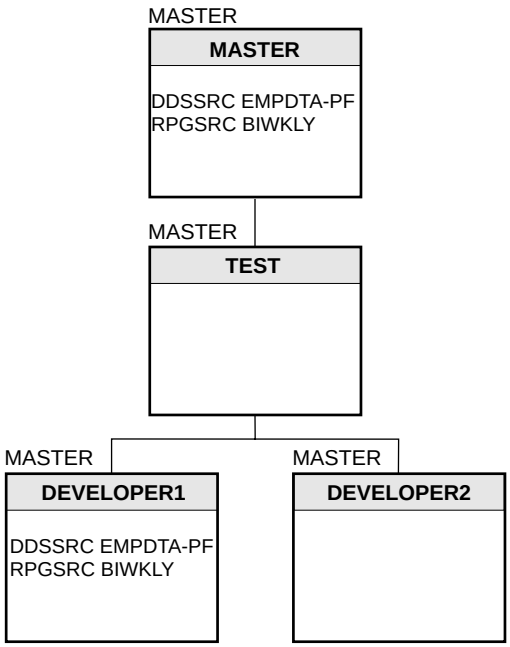

Figure 45. The Build Process and the Search Path

Parts exist in the groups MASTER and DEVELOPER1. DDSSRC EMPDTA-PF and RPGSRC BIWKLY are checked out to the group DEVELOPER1 so that they can be changed, but they still exist in the MASTER group. After they are changed, the BLDPART command is issued from group DEVELOPER1 with a build scope of *NORMAL and a search path of *DFT.

After a successful build, the following parts exist in the group DEVELOPER1:

DDSSRC EMPDTA-PF
FILE EMPDTA-PF
RPGSRC BIWKLY
PGM BIWKLY

Now suppose a developer in the group DEVELOPER2 requests to build the part RPGSRC BIWKLY. The group DEVELOPER2 does not contain any parts. The build process searches this group for the parts to build. The parts are not found in DEVELOPER2, so the build process looks in the group above DEVELOPER2 in the project hierarchy, the group TEST. This group contains no parts. The parts are built from the group MASTER, and the outputs PGM BIWKLY and FILE EMPDTA-PF are put in the group DEVELOPER2. This means that the developer in the group DEVELOPER2 has a version of the built parts that is different from the version the developer in the group DEVELOPER1 built.

You might want to create your own search path part if you want the build process to search for parts through a sequence of groups that is different from the default search path. Only one search path part is used for each BLDPART command that is issued. This part has a part type of SCHPTH. The SCHPTH parameter on the BLDPART command has two values: *DFT and *name*. They are described in “How Search-Path Parts Are Processed” on page 116.

Because the Application Development Manager/400 feature manipulates the AS/400 library list when the BLDPART command is run, *you should not specify library names in your source code when you are referencing dependency parts.* If you do, the compiler may not use the correct version of the part when it is searching through the AS/400 library list as defined in the search path. This may cause the BLDPART command to issue a warning message.

If an include or file part you are building does not exist in your project but exists in the AS/400 library list, a warning is issued in the build report. However, sometimes the build proceeds and no warning is issued:

- C/400 include files in QCC/H and ILE C/400 include files in QCLE/H

  The build process ignores system include files. File names enclosed in angle brackets (<...>) designate system include files. File names enclosed in double quotation marks ("...") designate user include files.

- DB2/400 include files

  The build process ignores DB2 for OS/400 include statements specified as follows:

  ```
  EXEC SQL INCLUDE SQLCA
  EXEC SQL INCLUDE SQLDA
  ```

  These statements are not true includes, in the sense that the DB2 for OS/400 compiler does not read source from another member or source file.

- RPG/400 includes in QRPG/QIRGINC

The BLDPART command searches only the groups in the search path to identify the message files used by the compiler. A warning is issued if the MSGF is not found or is not a part.

Using the Build Part Command

Use the BLDPART command to build one part or an entire application. Both developers and administrators can use this command. Typically, project administrators or team leaders build applications, while developers build parts or components of the application. This section describes how both administrators and developers can control whether the build process occurs, and which parts are built according to the

values specified on the build scope, build mode, and build force parameters of the BLDPART command.

You must specify a project name and a group name on the PRJ and GRP parameters of the BLDPART command. You must have update access to this group. A part type and a part name must also be specified.

To indicate that you want to build a part of a certain type and name, enter a name on the PART parameter and a type on the TYPE parameter. You can specify *ALL to indicate you want to build all parts of all types. For example, if you enter TYPE(*ALL) PART(*ALL), all parts of all types are built. You can also specify a generic name for the type and part. See “Required Parameters” on page 40 for the use of the generic.

The PARTL parameter is where you specify if you want to add the build output part to the part-list part. Specifying PARTL(*PRV) allows you to use the name of the last part-list part that you used for this project, if you are a system administrator. However if you are a developer, this parameter identifies the name of the last part-list part that was used for the specified group. This value is not valid if you are running this command using the QSECOFR user profile. The default value of the PARTL parameter is *NONE, which indicates that the build output parts will not be added to a part-list part.

The part-list part must be specified if the group specified on the command was created with PARTLREQ(*YES).

If a part-list part is specified on the BLDPART command, then:

- The part-list part must exist in the default search path starting from the group in which the parts are built.
- The parts created during the build are added to the part-list part, if they do not already exist. (This may happen even if the command failed to build the part.)

Note

If you are using a part-list part to record all the parts that you have changed or created for a fix by specifying that part-list part on the PARTL parameter of the commands that you used to do the fix (such as CHGPART, CRTPART, and so on) you may want to consider using a different part-list part on the PARTL parameter of the BLDPART command. This is because the BLDPART command only adds the build OUTPUT parts (that is, non-source parts) to the part list. (For example, parts of type PGM and FILE.) That part-list can later be used to export the fix to a production library if you want to export only non-source parts.

Use the LANG parameter if you only want to build parts with a particular language. The default, LANG(*ALL), builds parts of all languages. An example where this parameter might be useful is for parts of type FILE. To build all print files, specify TYPE(DDSSRC), PART(*ALL) and LANG(PRTF).

If you are building a part of type CBLSRC with language CBL or SQLCBL, and if it contains more than one program, the BLDPART command compiles only the first program. The other programs are ignored, and a warning message is issued in the build report.

System/36 applications cannot be built using the BLDPART command because the System/36 compilers do not return information that the build process needs. For more information on how to compile System/36 objects, see “Building System/36 Applications” on page 244.

Example: This example illustrates how the BLDPART command builds all the parts in the group MASTER.

Using the BLDPART command

```
BLDPART PRJ(PAYROLL) GRP(MASTER) TYPE(*ALL) PART(*ALL)
```

Using the Programming Development Manager utility

Select option 14 (Build) on the Work with Parts Using PDM display.

The optional parameters and their default values used on the previous BLDPART command are as follows.

```
SCHPTH(*DFT) SCOPE(*NORMAL) FORCE(*NO) BLDMODE(*COND) SAVLST(*NO)
BINDSTEP(*YES) PARTL(*NONE)
```

The next sections explain the SCOPE, FORCE, BLDMODE, SAVLST, BINDSTEP, and PARTL(*NONE) parameters in more detail. For more explanation of the SCHPTH parameter, see “How Search-Path Parts Are Processed” on page 116.

Setting the Build Scope

The SCOPE parameter on the BLDPART command controls the extent or range of the parts that are processed during the build process. Build scope refers to how the relationships between parts are used to determine which parts are considered for building. Cross-project parts are never built, but the build process does update the relationships *to* and *from* cross-project parts it finds. There are four values from which you can choose on the SCOPE parameter to determine which parts are built: *NORMAL, *LIMITED, *DIRCHAIN, and *EXTENDED.

If nothing is specified on the BLDPART command, the default is to use a build scope of *NORMAL. This is the most typical scope to use when building a part. A normal build scope makes the part specified on the BLDPART command current. A part is **current** when it is built with the latest version of all the source and related parts used to create it. In contrast, a part is **stale** if the source and related parts have changed since the part was last built. In other words, if the build scope of *NORMAL is used, the part specified on the PART parameter and all other parts on which it depends, and on which they depend, and so on, are built.

For example, if you build part RPGSRC BIWKLY with a normal build scope, and this program includes a reference to a record in the part FILE BIWKLY, both parts are processed. Two parts, PGM BIWKLY and FILE BIWKLY, are created as a result of this one build command.

Developers should use this value when testing related parts in their development group. Administrators should use it when building the application in each group in the project hierarchy.

A build scope of *LIMITED causes only the part specified on the BLDPART command to be processed. The references to other dependencies in the part being built are ignored. Developers should use this build scope after creating a new part or changing an existing part substantially. A build scope of *LIMITED allows you to verify that the part works (in isolation) as expected.

A build scope of *DIRCHAIN (or direct chain) performs the same action as a build scope of *NORMAL, but also takes the build processing one step further. Any parts directly dependent on any parts being built are also built.

A build scope of *EXTENDED performs the same action as a build scope of *DIRCHAIN, but it takes the build processing one step further. Any parts directly or indirectly related to all the parts that are built are also built. For example, administrators or developers could use a build scope of *EXTENDED to build an application following a change made to a field in a field reference file.

Figure 46 illustrates the build scope function. DDSSRC EMPID-PF is the part around which the build scopes *LIMITED, *NORMAL, *DIRCHAIN, and *EXTENDED are described. The long vertical lines indicate the parts involved in the build process, according to the scope specified. The broken vertical lines indicate the parts involved in the build process might be built, according to the scope specified. SCOPE(*NORMAL) is the default. This example assumes all parts are stale. Note that either the source DDSSRC EMPID-PF or the output FILE EMPID-PF can be specified on the BLDPART command. Both give the same result.

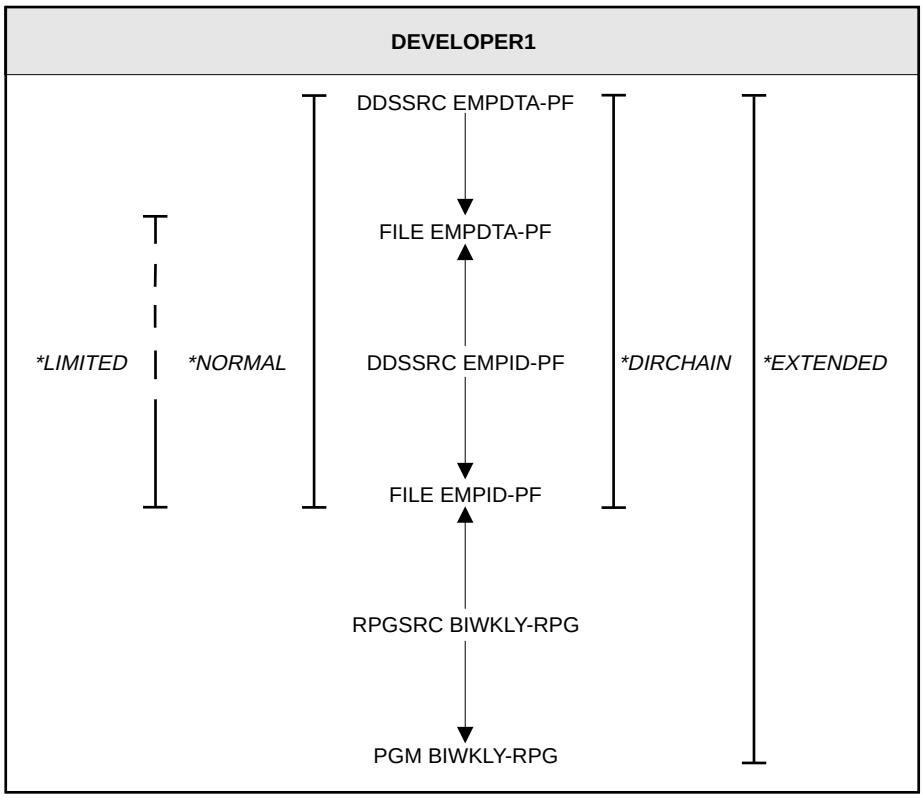

Figure 46. The BLDPART Command and the Build Scope of the DDSSRC EMPID-PF Part

If you choose a *LIMITED, *NORMAL, *DIRCHAIN, or *EXTENDED build scope, the parts in the preceding figure are built as follows:

*LIMITED
: Builds only DDSSRC EMPID-PF. FILE EMPID-PF is recreated, if FILE EMPDTA-PF exists. The relationships between DDSSRC EMPID-PF and FILE DDSSRC-LF and between DDSSRC EMPDTA-PF and FILE EMPDTA-PF are updated.

  All dependent parts are checked to see if any of them have changed, but they are not built. The *LIMITED build scope stops at the first level of dependencies when checking part dependencies for programs and files. The broken vertical line that is part of the *LIMITED build scope shows that the build process uses FILE EMPDTA-PF to determine if DDSSRC EMPID-PF needs to be built. If FILE EMPDTA-PF has changed, DDSSRC EMPID-PF is built.

*NORMAL
: The part that DDSSRC EMPID-PF depends on is built. In the example, it depends on FILE EMPDTA-PF, so DDSSRC EMPDTA-PF is built.

  The part specified on the BLDPART command, DDSSRC EMPID-PF, is built. FILE EMPID-PF is recreated.

*DIRCHAIN
: The same parts that were built for the *NORMAL build scope are built for the *DIRCHAIN build scope.

*EXTENDED
: The part that DDSSRC EMPID-PF depends on is built. In the example, it depends on FILE EMPDTA-PF, so DDSSRC EMPDTA-PF is built.

  The part specified on the BLDPART command, DDSSRC EMPID-PF, is built. FILE EMPID-PF is recreated.

  The part that depends on FILE EMPID-PF is built. It is RPGSRC BIWKLY. PGM BIWKLY is recreated.

  If you use SCOPE(*EXTENDED), all parts related to each other are checked and are built accordingly. Since all parts are inter-related in this example, specifying any part on the BLDPART command with SCOPE(*EXTENDED) achieves the same result.

If you build a part of type PGM, FILE, CLD, CMD MODULE, or SRVPGM, the BLDPART command compiles the source, if necessary, even if SCOPE(*LIMITED) is specified. The source or output part can be specified on the BLDPART command. The AS/400 object types supported by the Application Development Manager/400 feature are listed in Table 5 on page 61.

Using the same parts that were used in Figure 46 on page 149, let's build the compiled RPG program called BIWKLY. If you choose a *LIMITED, *NORMAL, *DIRCHAIN, or *EXTENDED build scope, the parts are built as follows:

*LIMITED
: Builds only PGM BIWKLY-RPG. We are requesting the BLDPART command to limit the build process to just PGM BIWKLY-RPG. RPGSRC BIWKLY-RPG is compiled.

  All parts that RPGSRC BIWKLY directly depends on are used to determine whether RPGSRC BIWKLY is compiled. In Figure 46 on page 149, this means that the build process uses FILE EMPID-PF to determine if RPGSRC BIWKLY-RPG needs to be built. If FILE

EMPID-PF or RPGSRC BIWKLY have changed, BIWKLY is built but FILE EMPID-PF is not compiled.

*NORMAL — The part that RPGSRC BIWKLY-RPG depends on is built. In the example, it depends on FILE EMPID-PF. Physical files are built before logical files; therefore DDSSRC EMPDTA-PF is built, and then DDSSRC EMPID-PF is built.

The part specified on the BLDPART command, RPGSRC BIWKLY-RPG, is built, and PGM BIWKLY-RPG is recreated.

*DIRCHAIN — The same parts that were built for the *NORMAL build scope are built for the *DIRCHAIN build scope.

*EXTENDED — The same parts that were built for the *NORMAL build scope are built for the *EXTENDED build scope.

Examples: The following four commands illustrate the use of the *LIMITED, *NORMAL, *DIRCHAIN, and *EXTENDED build scopes based on Figure 46 on page 149.

Based on the following command, only DDSSRC EMPID-PF is built.

```
BLDPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(DDSSRC) PART(EMPID)
        SCHPTH(*DFT) SCOPE(*LIMITED) FORCE(*NO)
        BLDMODE(*COND) SAVLST(*NO)
```

Based on the following command, DDSSRC EMPDTA-PF and DDSSRC EMPID-PF are built.

```
BLDPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(DDSSRC) PART(EMPID)
        SCHPTH(*DFT) SCOPE(*NORMAL) FORCE(*NO)
        BLDMODE(*COND) SAVLST(*NO)
```

Based on the following command, DDSSRC EMPDTA-PF and DDSSRC EMPID-PF are built. In this case, the same parts that were built for the *NORMAL build scope are built for the *DIRCHAIN build scope.

```
BLDPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(DDSSRC) PART(EMPID)
        SCHPTH(*DFT) SCOPE(*DIRCHAIN) FORCE(*NO)
        BLDMODE(*COND) SAVLST(*NO)
```

Based on the following command, DDSSRC EMPDTA-PF, DDSSRC EMPID-PF, and RPGSRC BIWKLY-RPG are built. RPGSRC BIWKLY-RPG will be built only if it was built before and it is now stale (so that a relationship is established with DDSSRC EMPID-PF).

```
BLDPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(DDSSRC) PART(EMPID)
        SCHPTH(*DFT) SCOPE(*EXTENDED) FORCE(*NO)
        BLDMODE(*COND) SAVLST(*NO)
```

Example: Figure 47 on page 152 illustrates the build scope function when a logical file called FILE MNTHLYL-LF is included. A field referenced in DDSSRC EMPID-PF is changed in DDSSRC EMPDTA-PF. DDSSRC EMPID-PF is the part around which the build scopes *LIMITED, *NORMAL, *DIRCHAIN, and *EXTENDED are described. The long vertical lines indicate the parts involved in the build process, according to the scope specified. The broken vertical lines indicate the parts involved in the build process which may not build, according to the scope specified. This example assumes all parts are stale, and these parts were built in the target

group before. Note that either the source DDSSRC EMPID-PF or the output FILE EMPID-PF can be specified on the BLDPART command. Both give the same result.

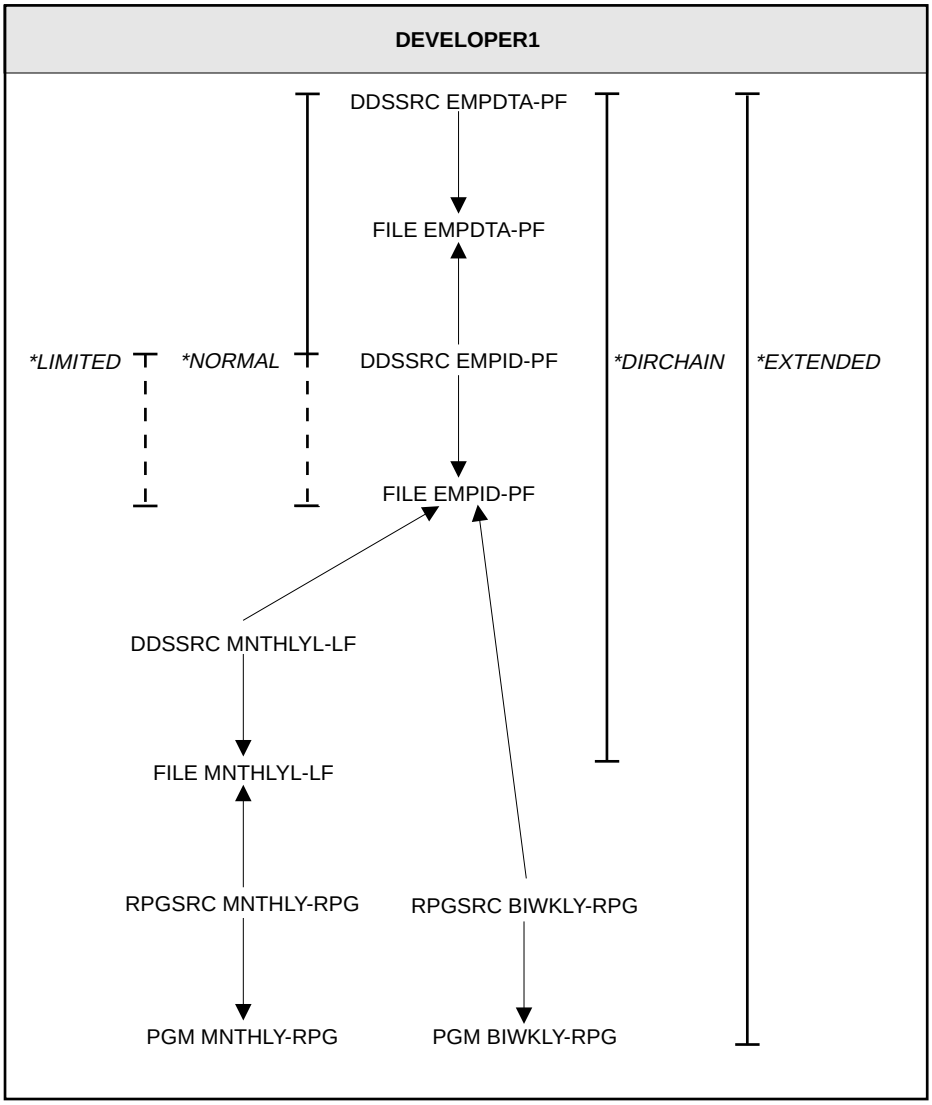

Figure 47. The BLDPART Command and the Build Scope of the DDSSRC EMPID-PF Part

If you choose a *LIMITED, *NORMAL, *DIRCHAIN, or *EXTENDED build scope, the parts in the preceding figure are built as follows:

*LIMITED — The broken vertical line indicates that DDSSRC EMPID-PF is not built and FILE EMPID-PF is not recreated, since the logical files are based on FILE EMPID-PF.

*NORMAL — The part that DSSSRC EMPID-PF depends on is built. In this example it depends on FILE EMPDTA-PF so DDSSRC EMPDTA-PF is built.

The part specified on the BLDPART command, DDSSRC EMPID-PF is not built, since the logical files are based on this file.

*DIRCHAIN The part that DSSSRC EMPID-PF depends on is built. In this example, it depends on FILE EMPDTA-PF so DDSSRC EMPDTA-PF is built.

The part specified on the BLDPART command, DDSSRC EMPID-PF is built.

The part that directly depends on FILE EMPID-PF is built. That means, DDSSRC MNTHLYL-LF is built. FILE MNTHLYL-LF is recreated.

*EXTENDED The part that DDSSRC EMPID-PF depends on is built. In the example, it depends on FILE EMPDTA-PF, so DDSSRC EMPDTA-PF is built.

The part specified on the BLDPART command, DDSSRC EMPID-PF, is built. FILE EMPID-PF is recreated.

The parts that depend on FILE EMPID-PF are built. DDSSRC MNTHLYL-LF is built to recreate FILE MNTHLYL-LF.

RPGSRC BIWKLY is built to recreate PGM BIWKLY.

If you use SCOPE(*EXTENDED), all parts related to each other are checked and are built accordingly. Since all parts are inter-related in this example, specifying any part on the BLDPART command with SCOPE(*EXTENDED) achieves the same result.

Determining Whether to Build All Parts

The build process uses a default of FORCE(*NO) when processing parts. This means that only those parts that need to be built are processed. This saves the build command from processing parts that do not need to be processed. If a part is current, it is not built again.

You can ensure that all parts are processed even if the part has not changed by specifying FORCE(*YES). You might want to do this to make certain that all parts in an application are updated.

Note that if you specify BLDMODE(*COND) and a dependent part has failed to build, any part that depends on it is not built, even when FORCE is *YES.

Suppose you have two parts in your group DEVELOPER1: RPGSRC BIWKLY, which has not changed since the last BLDPART command was issued and PGM BIWKLY, which is the output part for RPGSRC BIWKLY. If you choose to build RPGSRC BIWKLY with FORCE(*NO), nothing happens. BIWKLY is not compiled because it is current. If you chose to build it with FORCE(*YES), it compiles.

As parts are checked out, changed and promoted, parts in the group above the development level may quickly become out of date, possibly causing compiles of those parts to occur in the development group.

You may find that having the administrator periodically call BLDPART TYPE(*ALL) PART(*ALL) FORCE(*NO) in the group above the development groups, will significantly reduce the compiles done in the development groups.

Determining the Build Mode

The mode or method that the build process uses is determined by the value specified on the BLDMODE parameter. The three possible build modes are conditional, unconditional, and report only.

The default is BLDMODE(*COND). A conditional build mode means that when you specify a part on the BLDPART command, and if for any reason those parts that this part depends on fail to build, the part specified is not built. For example, part RPGSRC BIWKLY contains a reference to part FILE BIWKLY. If DDSSRC BIWKLY does not compile, neither PGM BIWKLY nor FILE BIWKLY is created.

A build mode of unconditional, BLDMODE(*UNCOND), indicates that even if a part does not compile successfully, parts dependent on it are still compiled. For example, if part DDSSRC BIWKLY fails to compile, RPGSRC BIWKLY will still be compiled.

A build mode of report only, BLDMODE(*RPTONLY), indicates that you do not want to actually compile parts but only produce a report listing the parts that *would* be compiled if *COND was specified on the BLDMODE parameter. Because this option prevents the compiling of parts, the build process uses the relationship information about parts from the last successful build only.

Saving the Build Report and Compiler Listings

A build report is always generated when you use the BLDPART command. Use the SAVLST parameter to save the build report and compiler listings into the file QALYBLDREP in the output group. The default is to not save them. To indicate that you do want to save them, specify SAVLST(*YES).

If you build a large number of parts and you want to automatically cleanup the spooled files generated by the BLDPART command, use the SAVLST(*DLT) parameter. This parameter will delete listings for all successful compiles. It will also delete the build reports if all the parts were successfully built, no warnings were issued and the BLDMODE(*RPTONLY) was not specified.

When a group is created, a physical file called QALYBLDREP is created in the new group. QALYBLDREP contains two types of output listings, build reports and compiler listings. Build reports are stored in members called QREPxxxxxx, where xxxxxx is replaced with a series of digits. The member type is REPORT. The description for these parts is composed of the name and type of the part that was specified on the BLDPART command.

The compiler listings are stored in members called QLSTxxxxxx, where xxxxxx is replaced by a series of digits. Their member type is the name of the report, QREPxxxxxx, as generated by the BLDPART command. The description for these members is composed of the name and type of the source part that was compiled.

Note: When SAVLST is *YES, the BLDPART command expects the compiler to generate the default spooled file name. The BLDPART command may not save the compiler listing if this is not the case. In some cases, this can be caused by changing the default name on the PRTFILE parameter of the compiler commands.

Figure 48 on page 155 shows the first page of a build report, which lists the parameters specified on the BLDPART command.

```
5716PW1   V3R7M0         Application Development Manager/400 - Build Report    08/11/96    12:00:00          Page . . :  0001


Project  . . . . . . . . . . . . . . . :  PAYROLL       1
Group  . . . . . . . . . . . . . . . . :  DEVELOPER1    2
Type . . . . . . . . . . . . . . . . . :  RPGSRC        3
Part . . . . . . . . . . . . . . . . . :  *ALL          4
Search path  . . . . . . . . . . . . . :  *DFT          5
Scope of build . . . . . . . . . . . . :  *NORMAL       6
Force build  . . . . . . . . . . . . . :  *NO           7
Build mode . . . . . . . . . . . . . . :  *RPTONLY      8
Save list  . . . . . . . . . . . . . . :  *NO           9
Perform bind step  . . . . . . . . . . :  *YES          10
Search path part used  . . . . . . . . :  *DFT          11
Search path used . . . . . . . . . . . :  PAYROLL                      DEVELOPER1
                                          PAYROLL                      TEST          12
```

Figure 48. The First Page of a Build Report—the Report Headings

1 The project specified on the BLDPART command.

2 The group specified on the BLDPART command.

3 The type specified on the BLDPART command.

4 The part specified on the BLDPART command.

5 The search path specified on the BLDPART command.

6 The scope specified on the BLDPART command.

7 The force specified on the BLDPART command.

8 The mode specified on the BLDPART command.

9 The SAVLST option specified on the BLDPART command.

10 The bind step options specified on the BLDPART command.

11 The search path part used by BLDPART (This is *DFT if the default search path was used.)

12 The search path used by BLDPART. (This can be a list of groups in a search path part, or it can be the default search path.)

Figure 49 shows the second page of a build report, which lists the messages, both the message number and message text, for warnings or errors found during the build process. See Appendix E, “Build Report Messages” for a list of the build report *reason for building* messages and a list of the build report *warning* messages that can occur.

```
5716PW1   V3R7M0         Application Development Manager/400 - Build Messages  11/08/96    12:00:00         Page . . :  0002

MSG ID    Text
-------   ------------------------------------------------------------------------------------------------------------
ADM4455   The compile for PAYROLL DEVELOPER1 RPGSRC BIWKLY failed.

                                  * * * * *  E N D  O F  B U I L D  M E S S A G E S  * * * * *
```

Figure 49. The Second Page of a Build Report—Messages

Figure 50 is the third page of the build report, which shows the build outputs created by the build process. The outputs are programs and files, for example, that are generated by compiling a source part. This section is categorized by source type. A table is printed for all the CSRC parts that were compiled, for all the logical files that were compiled, and so on. This format is used for the source types RPGSRC, CBLSRC, CSRC, CLESRC, CLDSRC, CMDSRC, DDSSRC (PF, LF, DSPF, PRTF, and ICFF), CLPSRC, PNLSRC, and BNDSRC.

```
5716PW1   V3R7M0          Application Development Manager/400 - Build Outputs   11/08/96    12:00:00              Page . . :  0003


      13                          15
      DDSSRC-PF                   FILE             BLDOPT
Part        Group                 Created     Part        Group                 Reason for Building  18
----------  --------------------  ----------  ----------  --------------------  ----------------------------------------------------
EMPDTA      DEVELOPER1            EMPDTA      QDFT        DEVELOPER1            Source part has not been built before.
 14                                 16          17


      DDSSRC-LF                   FILE             BLDOPT
Part        Group                 Created     Part        Group                 Reason for Building
----------  --------------------  ----------  ----------  --------------------  ----------------------------------------------------
EMPID       DEVELOPER1            EMPID       *DFT        *NONE                 Source part has not been built before.
                                       * * * * *  E N D  O F  B U I L D  R E P O R T * * * * *
```

Figure 50. The Third Page of a Build Report—Build Outputs

13 The type and language of the source part that was compiled; for example, DDSSRC-PF.

14 The name of the source part that was compiled and the group in which the part was found.

15 The type of output that was created, for example, program or file.

16 The name of the output part that was created.

17 The name of the BLDOPT part that was used, if any, and the group in which it was found. If the BLDOPT field in the report shows *DFT for part and *NONE for the group, as in the second line of the report in Figure 50, the default compiler command was used by the build process. No compiler command in a BLDOPT part was used to compile the part.

18 The reason the build process took place.

Binding a Program

If you want the build process to perform the Create Program (CRTPGM) command on a part of type MODULE, use the default on the *Perform bind step* parameter, BINDSTEP(*YES). If you do not want the bind step to be performed, specify BINDSTEP(*NO). Figure 51 on page 157 is the section of a sample build report that shows the outputs from the build process when BINDSTEP(*YES) is specified.

Figure 51 on page 157 shows the entry module source that was used, the BLDOPT part used, if any, and the reason the build was performed.

```
5716PW1   V3R7M0              Application Development Manager/400 - Build Outputs   11/08/96    17:49:11         Page . . :  0002

      CSRC                        OBJECT           BLDOPT
Part        Group                 Created    Part        Group                Reason for Building
----------  --------------------  ---------  ----------  -------------------  ------------------------------------------------
PGM2USRINC  USER      1           PGM2USRINC  *DFT       *NONE                *FORCE set to *YES.

      MODULE                      PGM              BLDOPT
Part        Group                 Created    Part        Group                Reason for Building
----------  --------------------  ---------  ----------  -------------------  ------------------------------------------------
PGM2USRINC  USER                  PGM2USRINC  PGM2USRINC  USER                Source part has not been built before.
                                   2           3                                4
                              * * * * *  E N D  O F  B U I L D  R E P O R T * * * * *
```

Figure 51. Sample Build Report that Shows the Outputs of the Bind Step

1. The source part and the object created
2. The part of type PGM that is created
3. The name of the build options parts that are used
4. The reason the part is built in each case

Adding Parts to the Part-List Part

The PARTL parameter is where you may add the parts to any part-list part. Specify the name of the part-list part or use the default, *NONE.

The PARTL parameter is required, if the specified group is defined with PARTLREQ(*YES). If the part-list part is specified, then:

- It must exist in the same promote path as the group in which the part is built
- The names of the output parts created during the build are added to the part-list part, if they do not exist

Building Fewer Parts

During peak times of development activity, you may want to build all the parts with SCOPE(*NORMAL) at least every night in a group separated from all other development groups, such as the integration and test group. All the build output parts such as FILEs, MODULEs, and PGMs in the integration and test group will then be up-to-date, and will not require rebuilding later. In addition, only parts affected by changes in the development group will get built normally, thereby improving the performance of the BLDPART command.

Using Build Options

This section discusses the following topics:

Things to consider before creating a build options part
Creating a build options part
How parts of type BLDOPT are processed
More than one build options part in the project hierarchy
Special build options commands for physical and logical files

Things to Consider Before Creating a Part of Type BLDOPT

The project administrator needs to determine a strategy for using parts of type BLDOPT. The project administrator should consider the following questions:

- Does your organization use only the compiler defaults when processing programs?

  If this is the case, you do not need to create parts of type BLDOPT for your application except for parts of type PGM (with languages CLE, RPGLE, CBLLE, and CLLE).

- Does your organization want to ensure that compiler options are used consistently across the application in the project hierarchy? For example, does your organization require that all programs be optimized when compiled?

- Does your organization want to ensure that compiler options are used consistently in each development group, in each test group, and in the root group?

  For example, in development groups you could: create parts of type BLDOPT that ensure programs are compiled with debug information; generate compiler listings; and include second-level message text in the compiler listing. In test groups, you could create parts of type BLDOPT that ensure programs are compiled without debug information, and generate compiler listings. In the root group, you could create a part of type BLDOPT to ensure that programs are compiled without debug information or without generating a compiler listing, and are optimized.

- Is the special value QDFT to be used as the name for a part of type BLDOPT?

  See “How Parts of Type BLDOPT are Processed” on page 165 for an explanation of how a build options part with this name is used. You should use this name for any build options part that contains the default compile options you want your organization to use by default.

Creating a Build Options Part

The BLDPART command uses all the default values associated with each optional parameter on the compiler commands when parts are processed, except for the CRTCMOD and CRTBNDC commands. The *Compiler options* parameter (OPTION) is set to the value *SYSINCPATH so that when you build an include part, the lowest version of the include in that particular branch of the project hierarchy is built. Note that *SYSINCPATH is not the default value for this parameter. The default value is *USRINCPATH. Figure 52 on page 161 shows the system-supplied build options part.

If you do not want the default values to be used when building a part, you must specify the exact command you want processed in a part of type BLDOPT. For example, you may want to specify *NOLIST on the CRTRPGPGM command so that no listing is generated.

The build options (BLDOPT) part is the Application Development Manager/400 part that allows you to specify compiler or processing commands. This part is created and managed in a similar way to other Application Development Manager/400 parts, except that you cannot build it.

To create a build options part, use the CRTPART command to create the part in the same way you create a source part. When you create a build options part, the CRTPART command automatically copies the default build options part from the QADM library installed with the Application Development Manager/400 feature into your part of type BLDOPT to give you a starting point.

This part contains all the compiler or preprocessor commands that the BLDPART command uses when processing parts. Each command specified in this part uses the default values for all its optional parameters, except the CRTCMOD and CRTBNDC commands described at the beginning of this section entitled “Creating a Build Options Part.”

The required parameters support substitution variables. This means that the appropriate library, source file name, and object name are derived from the part being built.

Each CL command in the default BLDOPT part is commented out. If you want to use a particular CL command, use the CHGPART command to edit the build options part you are creating. Remove the comment delimiters (/* and */) from the command you want to use, specify the compile options you want to use, and only delete any commands you do not need. You can have many compiler or preprocessor commands in one build options part.

When the build process uses the build options part, it does not check the syntax of the information found in the part. If there are errors in the syntax of the commands, the part fails to build. The build options part has an AS/400 member type of CLP; therefore, you should check the syntax of the part using the Source Entry Utility (SEU) to avoid specifying CL commands incorrectly. For more information on this utility, see the *ADTS/400: Source Entry Utility* book.

If you want to use your own preprocessor or post-processor, you can now use a label that is the same as the part type, and specify any command in the build option part. If a label with the same name as the part type of the part being built is found, then the CL command on that statement is run without any syntax checking. No other compiler commands are issued after running the labelled CL command. The substitution variables are supported on labelled CL command. If more than one valid commands are found for the part being built, the first labelled command is used. If a valid labelled command is found anywhere in the build options file, then it is used for building, even though another non-labelled compiler command precedes it.

Example: If you want to preprocess your RPGSRC part PART01 with your preprocessor called PRCRPGSRC, you can create a build option part PART01 and add a CL command as follows:

```
RPGSRC:  PRCRPGSRC OBJ(QTEMP/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)...
```

In this case, RPGSRC PART01 will be built using the PRCRPGSRC processor and not using the CRTRPGPGM command even though this CL command might have been uncommented. The PRCRPGSRC is a preprocessor, it should call the CRTRPGPGM command to create RPG programs from the intermediate source it may have generated. Since IBM compilers write Application Development Manager/400 API space and provide the build information, it is the responsibility of the preprocessor or the post-processor to call the appropriate compiler or to write the build information using APIs. For more information, see “Ensuring the Build Process Knows the Relationships Among Parts” on page 174.

If a label is found on a line by itself, then the command found on the next line will be used by the BLDPART command.

Note that in the build options part, the BLDPART command does not recognize commands that have labels other than the labels that are the same as the part types. Usually, you should not alter the example commands, other than to add new parameters. This ensures that the build process successfully runs the command. Figure 52 shows all the compiler and preprocessor commands that are copied into the build options part that you create.

```
/*********************************************************************/
/*                                                                   */
/* Application Development Manager/400 recognizes following          */
/* substitution variables in a build options part (part of           */
/* type BLDOPT):                                                     */
/*                                                                   */
/*                                                                   */
/* &F  - Source file name of the part being compiled.                */
/*                                                                   */
/* &L  - Library name associated with the group containing           */
/*       the part being compiled.                                    */
/*                                                                   */
/* &N  - Object/member name of the part being compiled.              */
/*                                                                   */
/* &O  - Name of the library where the output object of the          */
/*       BLDPART command is placed.                                  */
/*                                                                   */
/* &ZA - Language attribute of the source part being compiled,       */
/*       not of the output part.                                     */
/*                                                                   */
/* &ZC - &ZC is replaced with BLDPART.                               */
/*                                                                   */
/* &ZD - &ZD is replaced by the type of the output part created      */
/*       by the BLDPART command. If the source part has not          */
/*       been previously built, this is the same as the type         */
/*       of the source part.                                         */
/*                                                                   */
/* &ZE - &ZE is replaced by the name of the output part being        */
/*       created by the BLDPART command. If the source part has      */
/*       not been previously built, this is the same as the name     */
/*       of the source part.                                         */
/*                                                                   */
/*       For a user-defined type of *NONE, &ZE is set to the         */
/*       first output member met by the BLDPART command, or the      */
/*       name of the part with the user-defined type of *NONE,       */
/*       if the part has not been processed before.                  */
/*                                                                   */
/* &ZG - Name of the group where the outputs of the BLDPART          */
/*       command are placed.                                         */
/*                                                                   */
/* &ZN - Name of the part being compiled.                            */
/*                                                                   */
/* &ZO - The value of the Build Scope parameter on the BLDPART       */
/*       command. The values are *NORMAL, *LIMITED, *DIRCHAIN, and   */
/*       *EXTENDED.                                                  */
/*                                                                   */
/* &ZP - Name of the project containing the part being compiled.     */
/*                                                                   */
/* &ZS - The value of the Search Path parameter on the BLDPART       */
/*       command. This value is the name of the search path part.    */
/*                                                                   */
/* &ZT - Type of part being compiled.                                */
/*                                                                   */
/* &ZY - Name of the part-list part specified on the PARTL           */
/*       parameter.                                                  */
/*                                                                   */
/*********************************************************************/
         /*************************************************/
         /* PGM compilers                                 */
         /*************************************************/
         /* CRTRPGPGM  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                         REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE) */
         /* CRTCBLPGM  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                         REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE) */
         /* CRTCLPGM   PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                         USRPRF(*USER) REPLACE(*YES) AUT(*EXCLUDE) */
```

Figure 52 (Part 1 of 3). The System-Supplied Build Options Part

```
/*****************************************************/
/* FILE compilers                                    */
/*****************************************************/
/* CRTPF      FILE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               AUT(*EXCLUDE) */
/* CRTLF      FILE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               AUT(*EXCLUDE) */
/* CRTDSPF    FILE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               AUT(*EXCLUDE) REPLACE(*YES) */
/* CRTPRTF    FILE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               AUT(*EXCLUDE) REPLACE(*YES) */
/* CRTICFF    FILE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               AUT(*EXCLUDE) REPLACE(*YES) */
/*****************************************************/
/* SQL preprocessors                                 */
/*****************************************************/
/* CRTSQLRPG  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               OPTION(*GEN) REPLACE(*YES) */
/* CRTSQLCBL  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               OPTION(*GEN) REPLACE(*YES) */
/*****************************************************/
/* CLD/CMD compilers                                 */
/*****************************************************/
/* CRTCLD     CLD(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               AUT(*EXCLUDE) REPLACE(*YES) */
/* CRTCMD     CMD(&O/&ZE) PGM(&O/&ZE) SRCFILE(&L/&F) +
               SRCMBR(&ZN) AUT(*EXCLUDE) REPLACE(*YES) */
/*****************************************************/
/* ADDPFM/ADDLFM                                     */
/*****************************************************/
/* ADDPFM FILE(&O/&ZE) MBR(MBRNAME) */
/* ADDLFM FILE(&O/&ZE) MBR(MBRNAME) */
/*****************************************************/
/* UIM                                               */
/*****************************************************/
/* CRTPNLGRP  PNLGRP(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
               AUT(*EXCLUDE) REPLACE(*YES) */
/* CRTMNU     MENU(&O/&ZE) TYPE(*UIM) SRCFILE(&L/&F) +
               SRCMBR(&ZN) AUT(*EXCLUDE) REPLACE(*YES) */
/*****************************************************/
/* ILE compilers and preprocessors                   */
/*****************************************************/
/* CRTPGM     PGM(&O/&ZE) MODULE(&L/&ZN) REPLACE(*YES)   +
               USRPRF(*USER) AUT(*EXCLUDE) */
/* CRTSRVPGM  SRVPGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)  +
               REPLACE(*YES) AUT(*EXCLUDE) */
/* CRTCMOD    MODULE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)  +
               REPLACE(*YES) AUT(*EXCLUDE)              +
               OPTION(*SYSINCPATH) */
/* CRTSQLCI   OBJ(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
               OBJTYPE(*MODULE) OPTION(*GEN) REPLACE(*YES)*/
/* CRTBNDC    PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
               REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE)  +
               OPTION(*SYSINCPATH) */
/* CRTRPGMOD  MODULE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)  +
               REPLACE(*YES) AUT(*EXCLUDE)                */
/* CRTSQLRPGI OBJ(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
               OBJTYPE(*MODULE) OPTION(*GEN) REPLACE(*YES)*/
/* CRTBNDRPG  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
               REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE)  */
/* CRTCBLMOD  MODULE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)  +
               REPLACE(*YES) AUT(*EXCLUDE)                */
/* CRTSQLCBLI OBJ(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
               OBJTYPE(*MODULE) OPTION(*GEN) REPLACE(*YES)*/
/* CRTBNDCBL  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
               REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE)  */
/* CRTCLMOD   MODULE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)  +
               REPLACE(*YES) AUT(*EXCLUDE)                */
/* CRTBNDCL   PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
               REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE)  */
```

Figure 52 (Part 2 of 3). The System-Supplied Build Options Part

```
/*****************************************************/
/* CoOperative Development Environment/400 Compilers   */
/*****************************************************/
/* CRTRPGPGM  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE) +
                OPTION(*SRCDBG) */
/* CRTSQLRPG  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                OPTION(*GEN *LSTDBG) +
                REPLACE(*YES) */
/* CRTCBLPGM  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE) +
                OPTION(*SRCDBG) */
/* CRTSQLCBL  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                OPTION(*GEN *LSTDBG) REPLACE(*YES) */
/* CRTCLPGM   PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                USRPRF(*USER) REPLACE(*YES) AUT(*EXCLUDE) +
                OPTION(*SRCDBG) */
/* CRTCMOD    MODULE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                OPTION(*SYSINCPATH *EVENTF) +
                REPLACE(*YES) AUT(*EXCLUDE) +
                DBGVIEW(*SOURCE) */
/* CRTSQLCI   OBJ(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
                OBJTYPE(*MODULE) OPTION(*GEN *EVENTF) +
                REPLACE(*YES) DBGVIEW(*SOURCE) */
/* CRTBNDC    PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
                REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE)  +
                OPTION(*SYSINCPATH *EVENTF) DBGVIEW(*SOURCE) */
/* CRTRPGMOD  MODULE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)  +
                REPLACE(*YES) AUT(*EXCLUDE) +
                OPTION(*EVENTF) DBGVIEW(*SOURCE) */
/* CRTSQLRPGI OBJ(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
                OBJTYPE(*MODULE) OPTION(*GEN *EVENTF) +
                REPLACE(*YES) DBGVIEW(*SOURCE) */
/* CRTBNDRPG  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
                REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE) +
                OPTION(*EVENTF) DBGVIEW(*SOURCE) */
/* CRTCBLMOD  MODULE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)  +
                REPLACE(*YES) AUT(*EXCLUDE) +
                OPTION(*EVENTF) DBGVIEW(*SOURCE) */
/* CRTSQLCBLI OBJ(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
                OBJTYPE(*MODULE) OPTION(*GEN *EVENTF) +
                REPLACE(*YES) DBGVIEW(*SOURCE) */
/* CRTBNDCBL  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
                REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE) +
                OPTION(*EVENTF) DBGVIEW(*SOURCE) */
/* CRTCLMOD   MODULE(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)  +
                REPLACE(*YES) AUT(*EXCLUDE) +
                OPTION(*EVENTF) DBGVIEW(*SOURCE) */
/* CRTBNDCL   PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN)     +
                REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE) +
                OPTION(*EVENTF) DBGVIEW(*SOURCE) */
```

Figure 52 (Part 3 of 3). The System-Supplied Build Options Part

Note: The *Compiler options* parameter (OPTION) on the CRTBNDC and the CRTCMOD commands, is set to the value *SYSINCPATH so that when you build an include part, the lowest version of the include in the search path is built. Note that *SYSINCPATH is not the default value for this parameter. The default value is *USRINCPATH.

The substitution variables in the build options part resolve to the following:

&O The name of the library where the outputs of the build process are placed. This is the library for the project and group specified on the BLDPART command. &O should never be changed.

&ZE The name of the output part being created. If the source has not been previously built, this is the same as the source part name. &ZE can be changed if necessary.

The &ZE substitution variable resolves at build time. You can replace the &ZE variable with any of the following.

*PGMID, for COBOL/400 only
*CTLSPEC, for RPG/400 only
name

If possible, files, modules, and service programs should have the same name as the source part that creates them.

The BLDPART command automatically deletes an output part, for example CLD A, if the source part now compiles into an output part called CLD B in the same output group.

&L The name of the AS/400 library for the group that contains the source part. For example, you can issue the following command:

```
BLDPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
```

If the part RPGSRC BIWKLY is not in your group DEVELOPER1, but in a group called TEST that is one level higher in the project hierarchy, &L resolves to the TEST group's library. This is in contrast with &O, the library where the build outputs are placed. &L should never be changed.

&F The name of the file that contains the source part being compiled. &F should never be changed.

&ZN The name of the source part being compiled. &ZN should never be changed, except possibly for the PGM in CRTCMD.

For a complete list of substitution variables available in the Application Development Manager/400 feature, see Appendix D, “Substitution Variables.”

Using the BLDOPT Part for CODE/400

If you have the CoOperative Development Environment/400 (CODE/400) product installed, building Application Development Manager/400 parts using CODE/400 and you want to get the error feed back to CODE/400 after an Application Development Manager/400 build, then take the following steps:

1. Create an Application Development Manager/400 part of type BLDOPT with the appropriate name as discussed in the “Creating a Build Options Part” on page 158.
2. Ensure that this part is in the search path.
3. Uncomment the appropriate compiler command from the CODE/400 section of the system-supplied BLDOPT part as shown in Figure 52 on page 161.

How Parts of Type BLDOPT are Processed

Because you do not specify on the BLDPART command which BLDOPT part you want to use, there is a set search order used by the build command. The search order in which the BLDOPT parts are processed is:

1. The build process searches for BLDOPT *part-name*, where *part-name* is the name of the part being built, and uses the options found inside. For example, if RPGSRC BIWKLY is being built, the build process searches for BLDOPT BIWKLY.
2. If BLDOPT *part-name* is not found, the build process searches for BLDOPT *compiler-name*, where *compiler-name* is the name of the default compiler command to be used to build the part being built, and uses the options found inside. See Table 12 on page 141 for the names of the default compiler commands. Note that if more than one compiler commands are valid for the part being built, only the BLDOPT part with the default compiler command name will be recognized by the build process. However, you may remove the comments from the other compiler command, so the build process uses it. For example, if you are building RPGLESRC part, both CRTBNDRPG and CRTRPGMOD compiler commands are valid. However, the build process will only recognize the BLDOPT CRTBNDRPG. You may create the BLDOPT CRTBNDRPG part, but remove the comments from the CRTRPGMOD command in it.
3. If BLDOPT *compiler-name* is not found, the build process searches for part BLDOPT QDFT. This part is the default build-options part.
4. If BLDOPT QDFT is not found, the build process uses the default compiler options for the compile command that is being used. For example, if the source part type being compiled is DDSSRC-PF, the BLDPART command looks for the CRTPF command in a BLDOPT part in the order described in the preceding list. If it is not found, the default compile command is used. If there are many CRTPF commands in a BLDOPT part, the first one is used.

This means that the person who defines the build processing must be aware of what build options parts are present in the project hierarchy, and ensure that the correct build processing is performed. Some additional points to remember about BLDOPT parts are:

- BLDOPT parts in shared projects in the search path are ignored.
- You must specify the same module on the ENTMOD parameter of the CRTPGM command, as that specified on the CRTxxxMOD command (where xxx is the compiler language, such as C, CL, CBL and RPG).

  If you are using the BLDOPT part with the same name as the MODULE part, then you must specify that name on the ENTMOD parameter of the CRTPGM command.
- The value *ALL or *generic** is not allowed on the MODULE parameter on the CRTPGM and CRTSRVPGM commands.
- If more than one compiler command is used from the part BLDOPT QDFT, and this part is changed, this will cause the rebuilding of all the parts that were built using any of the commands in the BLDOPT QDFT part. Only use the part BLDOPT QDFT if you want to change the default compiler commands used by the BLDPART command. You may want to use the BLDOPT *compiler-name* instead.

- When processing a BLDOPT part, the BLDPART command runs the first command it finds for the type of the part it is processing, unless a command with a label the same as its part type is found anywhere in the BLDOPT part. For example, when processing a BLDOPT part to build a CSRC part, the BLDPART command runs the first CRTBNDC or CRTCMOD command it finds. If neither a CRTBNDC nor a CRTCMOD command is found, the default compile command CRTCMOD is used for parts of type CSRC.

Only parts of type BLDOPT that are called QDFT, or have the name of the default compiler commands, or have the same name as a part are used by the BLDPART command. For example, if you create a part BLDOPT MASTER, and there is no source part called MASTER in the project hierarchy, this build options part is never used.

If you are building a part of type MENU, the BLDPART command searches for the CRTMNU command in the build options part. The part must be defined as CRTMNU TYPE(*UIM).

More than One Build Options Part in the Project Hierarchy

The project hierarchy shown in Figure 53 has two parts of type RPGSRC and three parts of type BLDOPT. Each of the source parts has a build options part associated with it. In addition to the part-specific build options part, there is the default build options part called QDFT.

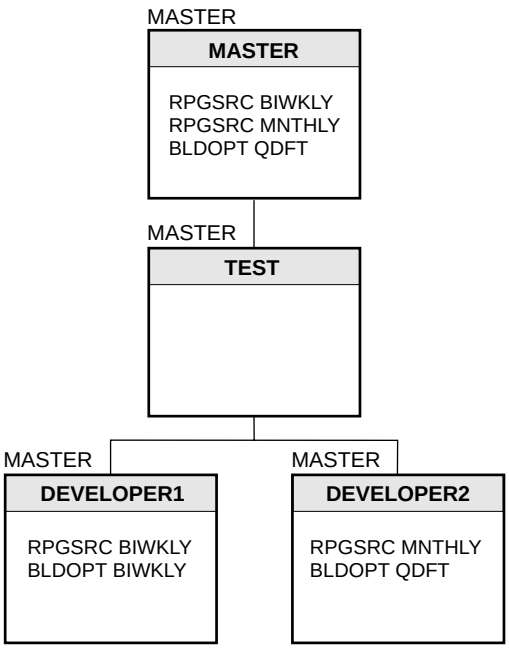

Figure 53. The Payroll Project with Parts of Type BLDOPT

The project administrator uses the following BLDPART command in each group.

```
BLDPART PRJ(PAYROLL) GRP(group-name) TYPE(RPGSRC) PART(*ALL)
```

Table 13 on page 167 identifies the build options part used to process the two RPGSRC parts.

Table 13. Build Option Parts Used to Process Two RPGSRC Parts

Group	Part	BLDOPT part used
MASTER	RPGSRC BIWKLY	PAYROLL MASTER BLDOPT QDFT
MASTER	RPGSRC MNTHLY	PAYROLL MASTER BLDOPT QDFT
TEST	RPGSRC BIWKLY	PAYROLL MASTER BLDOPT QDFT
TEST	RPGSRC MNTHLY	PAYROLL MASTER BLDOPT QDFT
DEVELOPER1	RPGSRC BIWKLY	PAYROLL DEVELOPER1 BLDOPT BIWKLY
DEVELOPER1	RPGSRC MNTHLY	PAYROLL MASTER BLDOPT QDFT
DEVELOPER2	RPGSRC BIWKLY	PAYROLL DEVELOPER2 BLDOPT QDFT
DEVELOPER2	RPGSRC MNTHLY	PAYROLL DEVELOPER2 BLDOPT QDFT

Note that the specification of how the CRTRPGPGM command is processed can be different in each of the parts of type BLDOPT. The search path starts with the group in which you are building. If a part of type BLDOPT is found there that matches the part you are building, that BLDOPT part is used.

To ensure consistency across the development groups, the project administrator can create a BLDOPT QDFT part with a promote code of *NONE in each development group. Similarly, a different version of BLDOPT QDFT could reside at the TEST and MASTER groups to ensure that build operations across the test or master levels are consistent. Figure 54 provides an example of this.

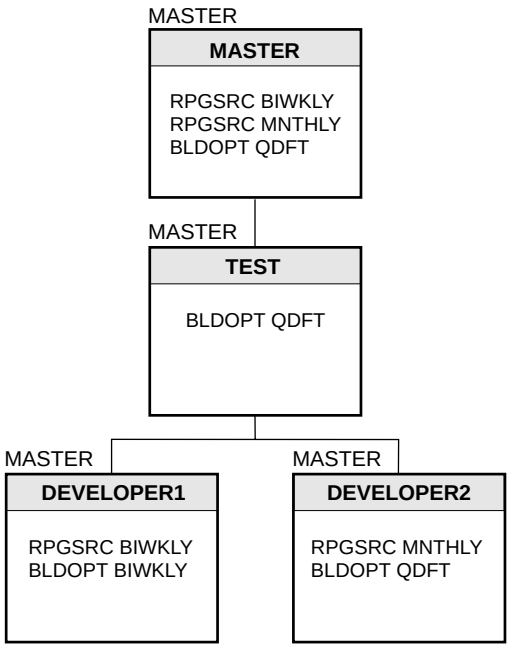

Figure 54. The Payroll Project with Parts of Type BLDOPT

In this figure there are three versions of the BLDOPT QDFT part, and each indicates the processing information that is required at each level in the project hierarchy.

If a developer needs to change the processing information, he or she creates a part-specific BLDOPT part or changes his or her copy of the QDFT build options part. The changed QDFT part or the *part-name* BLDOPT part should not be promoted to the TEST group.

Special Build Options Commands for Physical and Logical Files

Physical and logical files can have one or more members added to them, either through the Create Logical File (CRTLF) and the Create Physical File (CRTPF) commands, or through the Add Logical File Member (ADDLFM) and the Add Physical File Member (ADDPFM) commands. The build process searches the BLDOPT part for these commands, and runs the ADDPFM or ADDLFM commands after the physical or logical file has been successfully compiled. For example, to have the build process create a physical file with three members called simply A, B, and C, you can create a BLDOPT part to contain the following commands:

```
CRTPF FILE(&O/&ZE) SRCMBR(&ZN) MBR(A) MAXMBRS(3) ADDPFM FILE(&O/&ZE) MBR(B)
ADDPFM FILE(&O/&ZE) MBR(C)
```

There are five possible ways to define these commands in a BLDOPT part that affect how the BLDOPT part is processed, as described in "How Parts of Type BLDOPT are Processed." The following list describes the ways to specify the CRTPF and ADDPFM commands. You specify the CRTLF and ADDLFM commands in the same way:

1. The ADDPFM command is not defined in a BLDOPT part. The CRTPF command is not defined in a BLDOPT part.

   The default compile options for the CRTPF command are used.

2. The ADDPFM command is not defined. The CRTPF command is defined.

   Only the CRTPF command is used.

3. One or more ADDPFM commands are defined. The CRTPF command is not defined.

   The build process uses the default compile options for the CRTPF command, and performs the ADDPFM command after the CRTPF command, if the part compiles successfully.

4. One or more ADDPFM commands are defined. The CRTPF command is defined.

   The build process uses the CRTPF command and performs the ADDPFM command after the CRTPF command, if the part compiles successfully. If both the CRTPF and ADDPFM commands are defined, then they must exist in the same BLDOPT part.

5. A command is defined with the label DDSSRC.

Specifying OVRDBF and DLTOVR Commands

In the BLDOPT part, you can specify one or more Override Database File (OVRDBF) commands and one or more Delete Override (DLTOVR) commands to be issued during the build process. These commands allow you to set up certain environments or perform certain functions before and after a part is compiled.

The OVRDBF and DLTOVR commands that you want the BLDPART command to run must be in the same BLDOPT part as the compile command. All the OVRDBF commands in the part are run immediately before compiling the source part, and all the DLTOVR commands are run immediately after. The DLTOVR commands are run before the ADDPFM and ADDLFM commands are run, if ADDPFM and ADDLFM are specified.

The DLTOVR commands are intended to turn off the OVRDBF commands. If you do not match up the commands, the building of other parts could be affected.

These commands can contain substitution variables. The build process resolves the variables before running the command. Refer to Appendix D, “Substitution Variables” for a list of substitution variables.

If any OVRDBF command fails, the remaining OVRDBF commands that have not yet been run are not run. The part is considered to have failed. The DLTOVR commands are run regardless of whether any OVRDBF command failed or whether the source part failed to compile.

Special Build Processing for Specific Part Types

The following sections describe special considerations for using a part list part for building physical and logical files, MODULE, and MENU, and DB2 for OS/400 parts.

Building Parts Using a Part-List Part

When you use a part-list part with the BLDPART command, the part-list part is processed in the following way:

1. The search path specified on the BLDPART command is used to search for the part-list part.
2. Each part in the list is built, and any other parts, according to what you specify on the SCOPE parameter. If the part-list part is not found, the BLDPART command fails.

Building Physical and Logical Files

When you build a logical file, the physical file on which that logical file is based must be in the same group. If the physical file is not in the same group but is in the search path, it is copied into the build output group by the BLDPART command before compiling the logical file. If the physical file is not in the search path, the logical file fails to build.

When you want to build a physical file, the associated logical file must be in the build scope, or the physical file is not built.

The BLDPART command automatically deletes the object for the physical or logical file before calling the compiler. If the compile fails to re-create an object, the output part is also deleted.

Saving Physical File Data

If the SAVDTA parameter is set to *YES on the Create Project (CRTPRJ) command, the data in the physical file is saved before re-creating the file. After the compile is complete, the data is returned regardless of the success of the compile. This ensures that if the compile of your physical file part does not complete successfully, you do not lose your data.

If the data cannot be copied back to the file, you can locate the data that was contained in the physical file and copy it back into your physical file. The data from the physical file can be found in a part called QDTAxxxxxx in the output group, where xxxxxx is replaced by a series of digits. You should copy the QDTAxxxxxx *FILE back to the physical file part and rebuild that physical file part. The text description field of the QDTAxxxxxx part contains the part name of the physical file that was saved.

Parts of Type MODULE and the Build Process

The build process only performs the CRTPGM command when a MODULE part is successfully built, and a BLDOPT part with the same name as the MODULE part contains a CRTPGM command. The value specified on the BINDSTEP parameter must be *YES.

If you are binding one module, you must have a BLDOPT part with the same name as the module part, and you must specify BINDSTEP(*YES), or use the CRTBNDxxx (where xxx is the language, such as C, CL, CBL, and RPG). To bind several modules together, a BLDOPT part containing the CRTPGM must exist with the same name as the module part that is the entry point module.

If you are binding several modules together into a PGM or SRVPGM, there are a few different ways you can reference modules:

1. Specify modules on the MODULE parameter of the CRTPGM or CRTSRVPGM command in the BLDOPT part.
2. Reference a service program, part of type SRVPGM, which contains the referenced modules.
3. Reference modules through a bind directory part of type BNDDIR.

If you want to build the relationship between PGM and MODULE parts, then method 1 and 2 mentioned above are recommended. These methods provide the proper build relationship information, and the PGM or the SRVPGM is re-built, if any of the module changes.

If you successfully compile an ILE part using BINDSTEP(*YES), parts of type MODULE and PGM are created. Now, if you change the BLDOPT part to change the name of the MODULE part and re-build the ILE source part, a new part of type MODULE is created. The original MODULE part is considered as a stray object and is deleted. Note that to build the new MODULE part into a PGM, you must have a BLDOPT part with the same name as the new MODULE part containing the CRTPGM command.

Parts of Type CSRC, CINC and PGM with Language C and SQLC and the Build Process

The build part will not build parts of type CSRC, CINC and PGM with language C and SQLC.

In order to retain the data integrity, these parts will not be automatically converted to language CLE or SQLCLE. In order to change the language to CLE or SQLCLE, you can use the CHGPARTINF command.

Parts of Type MENU and the Build Process

When you build a part of type PNLSRC (language MENU) with the CRTMNU command, a part of type MENU is created. If you use a build options part, you must ensure that it contains the CRTMNU command and that the TYPE parameter on the CRTMNU command contains the value *UIM.

DB2 for OS/400 Parts and the Build Process

DB2 for OS/400 parts are source parts that contain DB2 for OS/400 statements and have a DB2 language attribute; for example, an RPGSRC part with the language attribute SQLRPG. The following source parts have an SQL language attribute:

Table 14. Application Development Manager/400 Source Type and Language

Source Type	Language
CBLINC, CBLSRC	SQLCBL
CBLLEINC, CBLLESRC	SQLCBLLE
CINC, CSRC	SQLC[1], SQLCLE
RPGINC, RPGSRC	SQLRPG
RPGLEINC, RPGLESRC	SQLRPGLE

Note:

1. The source part of type CSRC and CINC with language SQLC is not buildable.

The CRTSQLRPG, CRTSQLCBL, CRTSQLCI, CRTSQLRPGI, and CRTSQLCBLI commands have an OPTIONS parameter that can be set to either *GEN (the default) or *NOGEN. The Application Development Manager/400 feature only supports the *GEN option. When you specify *GEN, an DB2 for OS/400 temporary source file member is created, and the preprocessor calls the CRTRPGPGM, CRTCBLPGM, CRTBNDC, CRTCMOD, CRTBNDRPG, CRTRPGMOD, CRTBNDCBL, CRTCBLMOD, and CRTSRVPGM commands to create the program.

If a DB2 for OS/400 temporary source file member is the only object created as a result of using the *NOGEN option, the build process considers the part built, assuming the compilation of the part was successful. If you refer to files using DB2 for OS/400 statements, the BLDPART command may not automatically build these files since the DB2 for OS/400 language does not require these files to successfully preprocess the source code.

If a DB2 for OS/400 include part refers to a file, the source part that uses the include part may refer to that file as well. In this case, there would be an external reference relationship from the include part to the file and from the source part to the file.

Messages Displayed for Each Part Being Compiled

This section explains why the same message might be displayed more than once while the BLDPART command is running. These messages are displayed only if you are not running in batch mode.

Parts are built one after the other until all are built. Your display is updated with a message for each part that is being compiled. An example of some messages are:

```
Compiling CSRC BIWKLY...
Binding BIWKLY...
```

As the build process learns about new part relationships from the compiler, it may have to backtrack to build dependencies and recompile a part again. Thus, the same message might be displayed more than once. **Backtracking** is required when a part is changed in such a way that it involves another part that was not involved earlier in the build process.

Building a Sample Application

This section describes a series of build and promote commands for the hypothetical application called SAMPLE in Table 15.

Table 15. Parts for the SAMPLE Application

Group	Part	Part Type
MASTER	REFMST	DDSSRC
MASTER	CTLFIL	DDSSRC
MASTER	EMPMST	DDSSRC
MASTER	RSNMST	DDSSRC
MASTER	TRWEEK	DDSSRC
MASTER	TRWEEKL	DDSSRC
MASTER	PRG06RP	DDSSRC
MASTER	MSGS	MSGF
TEST	PRG03FM	DDSSRC
TEST	PROC3	CLPSRC
TEST	PRG03	RPGSRC
TEST	PRG03A	RPGINC

The parts of the application reside in the groups as illustrated in Figure 55 on page 173.

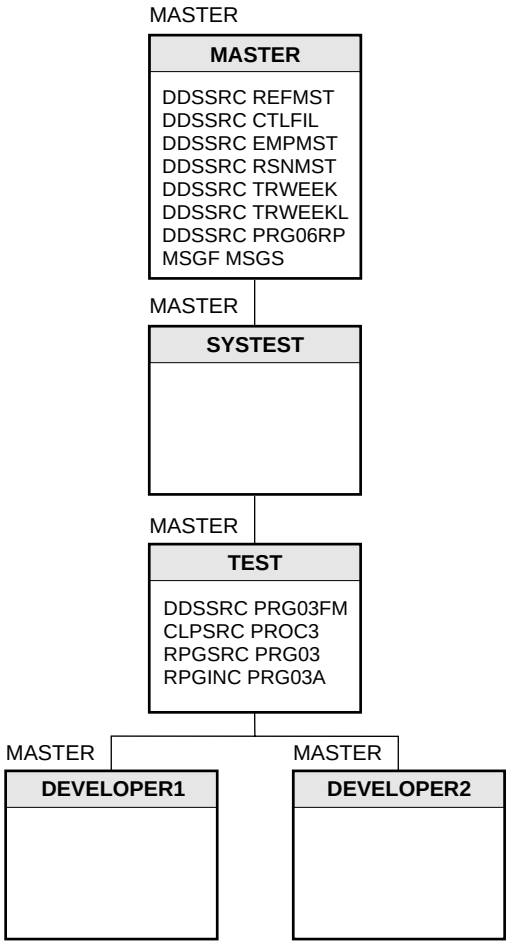

Figure 55. The Parts in the Sample Application

The following example identifies the steps to follow to promote and build parts in the sample application:

1. Promote all the parts from the TEST group to the SYSTEST group.
2. Build all parts of all types in the group SYSTEST.

   ```
   BLDPART PRJ(SAMPLE) GRP(SYSTEST) TYPE(*ALL) PART(*ALL)
   ```

   This build command processes all the parts found in SYSTEST using a normal build scope, a conditional build mode, and does not force current parts to be processed.
3. Examine the build report spooled to the output queue for this job and ensure that the parts were processed without any errors. If errors occurred, the developer responsible for the part should be notified and asked to correct the part or parts with errors. The Print Project Log (PRTPRJLOG) command can be used to determine who last changed the part. See “Using the Project Log” on page 208 for information on how to use this command.

   Build the parts again, and verify that the changes work.
4. Run the application to see if the parts work together as expected. See Chapter 12, “Testing and Running an Application” for information on how to do this.

5. Examine the build report to be sure that the parts were processed without any errors.
6. Run the application to verify that the parts work together as expected.
7. Promote all the parts to the group MASTER.

   ```
   PRMPART PRJ(SAMPLE) GRP(SYSTEST) TYPE(*ALL) PART(*ALL)
   ```

8. Build all the parts at the group MASTER.

   ```
   BLDPART PRJ(SAMPLE) GRP(MASTER) TYPE(*ALL) PART(*ALL)
   ```

9. Examine the build report to be sure that the parts were processed without any errors.
10. Run the application, and verify that it works as expected. If it does, the application is now ready to be exported to a production environment. See Chapter 13, “Exporting an Application” for information on how to do this.

The Build Process and User-Defined Types

If you have created a user-defined type that falls into one of the following two categories, you will want to understand how the build process handles a part of the new type:

1. User-defined type represented as a source member that you use with a compiler that does not support the Application Development Manager/400 feature.
2. User-defined type for a part that is not stored as an AS/400 object or source file member and, when processed, generates one or more source file members.

This section describes:

- What you need to do to ensure that the build process learns about part relationships
- Building a part that is not stored in an object or member
- Building a part stored in a source file member
- How to use build options parts with the new part types you have defined

Ensuring the Build Process Knows the Relationships Among Parts

For a compiler to support the Application Development Manager/400 feature, or if the new part type you have defined for your compiler generates an object or source file member when you run the BLDPART command, the compiler needs to create the object or member in the library of the output group that was specified on the BLDPART command. (You can specify the &L substitution variable on the BLDCMD parameter of the ADDADMLANG command to pass this value to your compiler.) You must use the following application programming interfaces (APIs) described in the *System API Programming* and *System API Reference* books:

- Get Space Status (QLYGETS)
- Read Build Information (QLYRDBI)
- Set Space Status (QLYSETS)
- Write Build Information (QLYWRTBI)

These APIs allow the Application Development Manager/400 feature to understand the relationships among parts so that the BLDPART command establishes these relationships correctly.

Building a Part that is Not Stored in an Object or Member

When you add a user-defined type with the ADDADMTYPE command, you can define the object type that is to store a part of the new type as SYSTYPE(*NONE). This means that a part of the new type is not stored in an AS/400 object or source file member in an Application Development Manager/400 project. Parts that are not stored in an AS/400 object or member are processed first when the BLDPART command is run. The outputs from these parts are source file members. If they can be built, they are processed by the BLDPART command after the parts that are not stored in AS/400 objects or members are processed.

The type and language of the source file member parts created by the build process are specified by the given command in the Application Development Manager/400 API.

Note

BLDPART does not delete stray members. To illustrate, assume that BLDPART builds a part that is not stored in an object or a member and the build output is a member. If BLDPART is called again to build the same part and the build output is a different member, the original build output becomes a stray. This stray is not removed by BLDPART.

Building a Part that is Stored in a Source File Member

The build process searches for external references to system-supplied types such as MSGF. However, it does *not* search for user-defined types that represent the object type, such as *MSGF. You might receive a warning message in the build report if the externally referenced object is not a system-supplied part type.

If you define the object type that is to store a part of the user-defined type as SYSTYPE(*MBR), this means that the parts are stored in an AS/400 source file member. If the parts are source parts, they are built by the BLDPART command and only *PGM objects can be generated. If the parts are include parts, they are not built because a build command cannot be defined for them.

If the *PGM object that is created does not have one of the associated languages that are listed in Table 2 on page 45, you need to add a new type for the *PGM object and a new language for the type.

Note: If a program object can have two or more parts associated with it, the second object might overwrite the first object.

Using a Build Options Part with a User-Defined Type

When the BLDPART command builds a part with a user-defined type, the build process searches for the command to build the part in the following order:

1. The build options part is searched for a command that begins with the first character string token that you entered on the BLDCMD parameter of the ADDADMLANG command.
2. If a matching command is found in the build options part, it is used to build the part. (The command string in the build options part can contain more parameters than the command string you specified on the BLDCMD parameter of the ADDADMLANG command.)
3. If a matching command is *not* found, the part is built with the command string that you specified on the ADDADMLANG command.

Example: The following commands illustrate the preceding explanation. Assume that you specify the following values on the BLDCMD parameter of the ADDADMLANG command.

```
BLDCMD('CRTPASPGM PGM(&L/&ZN) SRCFILE(&L/&F) SRCMBR(&ZN)')
```

However, the following command string is found in the build options part.

```
CRTPASPGM PGM(&L/&ZN) SRCFILE(&L/&F) SRCMBR(&ZN)
          TEXT('A sample Pascal program')
```

The command string found in the build options part is used because it begins with the same character string specified on the BLDCMD parameter of the ADDLANG command.

Customized Include Files and BLDPART Command

If you are using customized include source files and do not want the build information (relationship) saved for these files during the build, then you can keep them in those system libraries with names beginning with 'Q' or '#'. The include relationship for such include files and any part including them will not be saved when the part is built. For example, if your include file resides in QGPL, then the relationship to that include will not be saved.

Chapter 12. Testing and Running an Application

This chapter describes how to:

- Create a project hierarchy with the role of a *tester* in mind
- Test an application within the Application Development Manager/400 feature
  - Add project libraries to the AS/400 library list for testing purposes
  - Remove project libraries from the AS/400 library list after testing

If you cannot test your application within the Application Development Manager/400 feature, you can export it, or components of it to an AS/400 library. See “Exporting an Application to Test It” on page 183 for information on how to test an application outside the control of the Application Development Manager/400 feature.

Defining a Project Hierarchy with a Group for a Tester

An **application tester** is a person whose primary role is to run the application and verify its correctness. Your organization may have individuals assigned solely to this role, or application developers who perform the activities of a tester at certain points in the application development cycle.

Typically, a tester does not perform part development activities. A tester is usually only given read access to the project hierarchy, or is assigned a development group for which update access is granted. Both situations are described below.

Tester with Read Access: The tester is given read access to the entire project hierarchy, but is not given update access to any groups. The following command authorizes a tester to the project PAYROLL.

```
ADDPRJUSR PRJ(PAYROLL) USRPRF(JONES) USRTYPE(*DEVELOPER) ACCESS(*READ)
```

In this case, the tester must work outside the project hierarchy to have files or objects updated if the application updates files when it is run. For the application to be tested outside Application Development Manager/400 control, it must be exported. See “Exporting an Application to Test It” on page 183 for information on how to do this.

Tester with Update Access: The tester is given read access to the entire project hierarchy, and is given a development group to work in. The tester's development group should be different from a developer's development group in that the tester should not be able to promote parts. The following commands create the group TESTER1 in the project PAYROLL, and enroll JONES to the new test group.

```
CRTGRP PRJ(PAYROLL) GRP(TESTER1) SHORTGRP(TST1) PARENT(TEST) PRMCODE(*NONE)
       CCSID(*PARENT) TEXT('JONES DEVELOPMENT GROUP - TESTER1')

ADDPRJUSR PRJ(PAYROLL) USRPRF(JONES) USRTYPE(*DEVELOPER)
          ACCESS(*UPDATE) DEVELOPGRP(TESTER1)
```

The group TESTER1 is created with a promote code of *NONE, meaning that parts within it cannot be promoted to the next group. JONES can, however, check parts out, change parts, copy parts, and create new parts with promote codes of *NONE, and build the application from within this group.

Typically, testers create the files required for application updates during their testing. Parts that contain high-level-language programming statements are not usually created, copied, or changed.

The project hierarchy in Figure 56 illustrates one way of defining a project hierarchy with a development group for a tester.

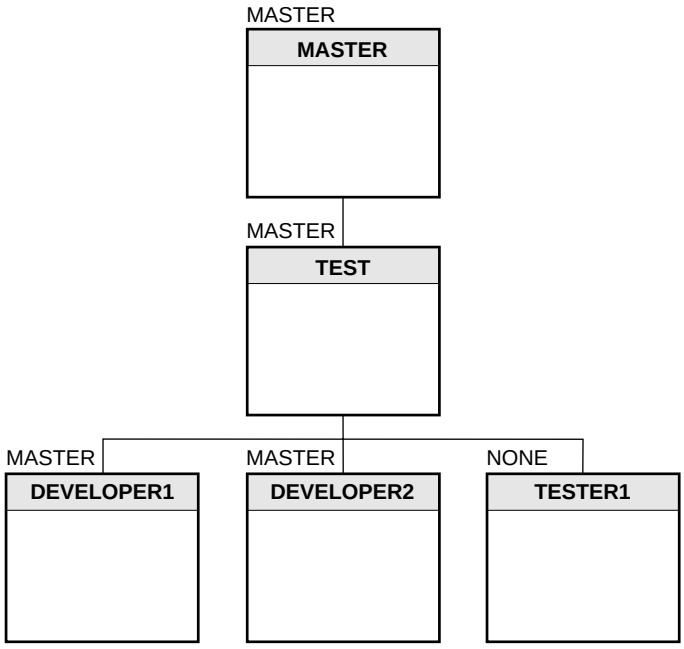

Figure 56. The Payroll Project with a Group for a Tester

The tester is able to test the parts that have been promoted to the group TEST by the developers working in the groups DEVELOPER1 and DEVELOPER2.

Note that the data files created and used to test the application in the group TESTER1 need to be up to date with respect to the part development activity going on at the same time. Errors could occur if the developer working in the group DEVELOPER1 changes the record format in a part, and then promotes the part to the group TEST. The data file created to test this record format must also be updated.

To avoid situations such as the one described above, the administrator could create a project hierarchy that stages the part development activity and gives the testers a more stable version of the application to work with. Such a project hierarchy is shown in Figure 57 on page 179.

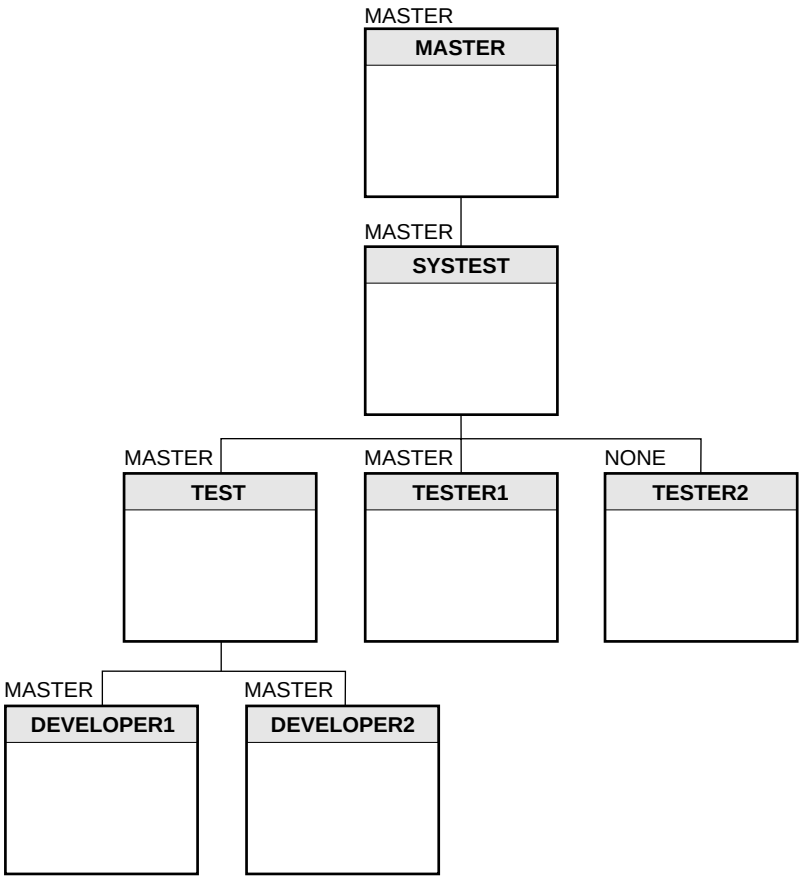

Figure 57. The Payroll Project with Test Stages

In this project hierarchy, the testers work at a higher level in the project hierarchy. Parts must be promoted from the development groups to the group TEST. Administrators then promote the parts to the group SYSTEST, and build the parts in that group. This way testing can occur on a consistent version of the application.

Testing a Part or an Application within the Project Hierarchy

To test a part or an application successfully with the Application Development Manager/400 feature, the underlying library list must be set up according to how the part or the application would work in a production environment. See “Groups and Their Relationship to the AS/400 Library List” on page 111 for an illustration of how groups correspond to the library list.

You do not need to manipulate the AS/400 library list, except when you want to test a part or an application. To do this, you need to use the Add Project Library List (ADDPRJLIBL) command to add all the project libraries that represent the search path to the user portion of your library list.

Adding Project Libraries to the Library List

Use the Add Project Library List (ADDPRJLIBL) command to add all the project libraries for a project's search path to the AS/400 library list. All the libraries associated with the search path indicated on the command are added to the user portion of the library list. Both developers and administrators can use this command.

You must specify a project name on the PRJ parameter. You must also specify a group name on the GRP parameter.

The SCAN parameter defaults to look for SCHPTH parts using the search path defined by the project hierarchy. The command starts from the group specified on the GRP parameter and searches up the default search path to the root group. If you choose SCAN(*NO), the ADDPRJLIBL command adds only the library for the group specified on the GRP parameter.

The SCHPTH parameter defaults to *DFT. If a part called SCHPTH QDFT exists in the default search path, it is used. If SCHPTH QDFT does not exist, the default search path is used to determine which project libraries to add the library list. You can also specify the name of a specific search-path part on the SCHPTH parameter. For example, you could create and use a SCHPTH part called COMMON to indicate that you want to add the project libraries that represent a cross-project search path to your library list.

If you enter the name of a search-path part on the SCHPTH parameter but specify SCAN(*NO), the search-path part name is ignored.

Example: Consider the project hierarchy illustrated in Figure 57 on page 179. The following ADDPRJLIBL command adds the project libraries for the search path defined in the part SCHPTH QDFT.

Using the ADDPRJLIBL command

```
ADDPRJLIBL PRJ(PAYROLL) GRP(TESTER1) SCAN(*YES) SCHPTH(*DFT)
```

Using the Programming Development Manager utility

Select option 45 (Add project library list) from the Work with Groups Using PDM display or use the user-defined option AP.

The part SCHPTH QDFT contains the following project and group combinations:

```
PAYROLL TESTER1
PAYROLL TEST
PAYROLL SYSTEST
PAYROLL MASTER
```

The library list is changed to include the project libraries representing the groups defined in the SCHPTH QDFT part, and the new library list is depicted in Figure 58 on page 181.

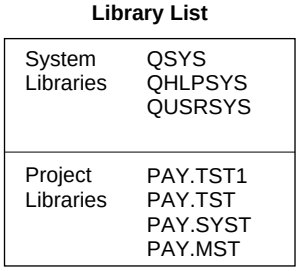

Figure 58. The New Library List

When you use the ADDPRJLIBL command, the system portion of the library list is not changed. The user portion is changed to include the Application Development Manager/400 project libraries. The current library, if set, remains the current library. Other libraries that were added to your library list are not removed. However, the project libraries are added to the top of this list. Refer to the next figure to see an example of how the library list appears before and after the ADDPRJLIBL command is run.

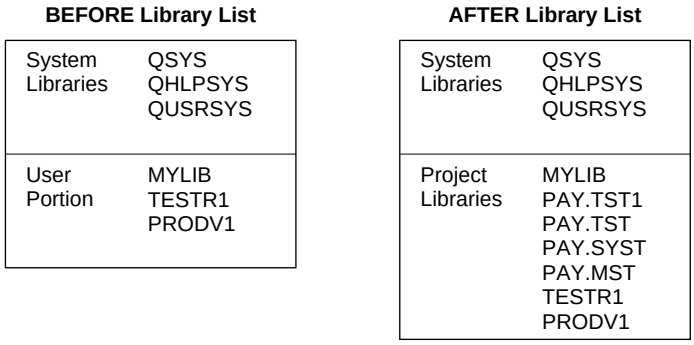

Figure 59. The Library List Before and After the ADDPRJLIBL Command is Run

In this figure, the current library is MYLIB. It remains as the current library after the ADDPRJLIBL command is entered. The libraries TESTR1 and PRODV1 are moved to the bottom of the library list.

The ADDPRJLIBL command is useful when you need to test object types that are not supported by the Application Development Manager/400 feature. For example, in Figure 59, an unsupported object type exists in the TESTR1 library. Your part needs to access the object when you build or run it. The ADDPRJLIBL command adds the project libraries to the library list and moves the libraries in the user portion to the bottom of the list. When you build or run your part, the object is found in the TESTR1 library.

The add project library list function is also available on the Work with Parts Using PDM display as Option 45 (Add project library list), or as the system-supplied user-defined option, AP. See “Adding and Removing Project Libraries” on page 238 for more information.

Adding External Libraries to the Library List

You can use the Add Project Library List (ADDPRJLIBL) command to add one or more external libraries to the AS/400 library list by updating the SCHPTH QDFT as follows. To do this, you must specify the search-path part containing the names of the external libraries.

In the following example, you will add external libraries to your library list. The search-path part contains the following information:

```
PAYROLL TESTER1
PAYROLL TEST
PAYROLL SYSTEST
PAYROLL MASTER
*USRLIB SOFTLIB1
*USRLIB SOFTLIB2
```

When you use the ADDPRJLIBL command with this search-path part, only the user portion of the library list changes.

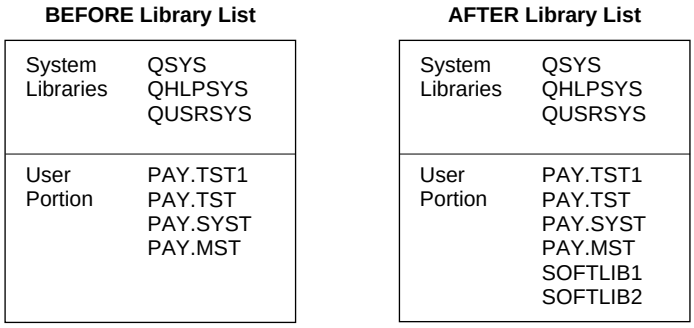

Figure 60. The Library List Before and After the ADDPRJLIBL Command is Run.

Figure 60 shows that after the ADDPRJLIBL command is issued, the SOFTLIB1 and SOFTLIB2 libraries are added to the user portion of the library list. It is added after the Application Development Manager/400 projects to ensure that the build searches for objects in the Application Development Manager/400 projects before the user libraries are searched.

Removing Project Libraries from the Library List

Use the Remove Project Library List (RMVPRJLIBL) command to remove *all* the project libraries added to the AS/400 library list with the ADDPRJLIBL command.

There are no parameters on the RMVPRJLIBL command. Simply enter the name of the command to remove all project libraries from your library list. Since there are no parameters on the RMVPRJLIBL command, the external libraries, which might be added by the ADDPRJLIBL command, are not removed.

The remove project library list function is also available from the following displays:

- The Work with Groups Using PDM display as function key F20=Remove project library list
- The Work with Parts Using PDM display as option 45

Chapter 13. Exporting an Application

This feature provides an **export** function through the EXPPART command. When you export, one or more parts of an application are copied from an Application Development Manager/400 project hierarchy into a given AS/400 library.

Note: You can move the Application Development Manager/400 project libraries and archive libraries to an alternate search path as long as the entire project resides in the same alternate search path.

You can export a single part or an entire version of an application. Typically, a developer would export a part or a component of an application for critical testing purposes, while an administrator or team leader would export an entire application for critical testing purposes or to copy a completed application to production.

This chapter describes:

- Exporting an application for testing
- Using the Export Part (EXPPART) command
- Packaging an application after you have exported it

In addition to using the EXPPART command to export parts to a production library or for testing purposes, you can also use it to copy parts to a library that you intend to restore on another AS/400 system. See “Copying Information to Another AS/400 System” on page 207 for information on how to use the EXPPART and IMPPART commands to do this.

You should take the time to read the section in the *ADTS/400: Application Development Manager Introduction and Planning Guide* that lists things to consider before moving to a production environment.

Exporting an Application to Test It

If any of the following conditions are true for your application, it may be necessary for you to export it, or components of it, to an AS/400 test library to test it:

- If the application must override files.
- If you need to test whether a certain user has the appropriate access to files, commands, or programs.
- If the application requires journaling or if the backup procedures are being tested. (Journals and journal receivers are AS/400 objects that are not directly supported by the Application Development Manager/400 feature.)
- If the part or application requires a unique library list, or if it changes the library list, it might need to be tested outside the project hierarchy. For example, if it uses commands that assign a product library, it cannot be tested within this product. It must be exported.
- If the application submits jobs to the batch subsystem while processing, the library list for these jobs could be taken from the job description rather than from the project hierarchy.

Using the Export Part Command

Note: Cross-project system-lists (that contain parts that reside in more than one project) must be exported with the CROSSPRJ(*YES) and SCAN(*YES) parameters. Also, the system-list and search path must exist in the project specified in the EXPPART command.

Use the Export Part (EXPPART) command to copy parts or components of your application from an Application Development Manager/400 group to an AS/400 library. When you export parts, a copy of the part is created in a library. You use your current AS/400 authorities to work with this command. The EXPPART command does not grant you additional authority.

In the library to which you are exporting, the part is copied to either a source file member or an object, depending on the part type. For example, if you are exporting a part of type RPGSRC, a source physical file is created if it does not already exist in the library, and a member within that source physical file is also created. If the source physical file already exists, the member is added to it. Source files are not replaced. If a physical data file already exists in the library, it cannot be replaced. Members and other objects that already exist in the library can be replaced.

As with the IMPPART command, when you export information you specify *from* and *to* criteria. The *from* criteria indicate what information you want to copy from the project hierarchy. The *to* criteria indicate where you want that Application Development Manager/400 information to be copied to.

Use the EXPPART command to export parts of a particular language. The default is LANG(*ALL).

The CHGDATE parameter of the EXPPART command allows you to export only parts changed since a specific date. CHGDATE(*BEGIN) is the default which exports all parts regardless of their last changed date. If you want to export all the parts that were changed today, specify CHGDATE(*CURRENT).

You can also export cross-project parts using the CROSSPRJ parameter. This parameter is ignored if SCAN(*NO) is specified or if no cross-project groups are in the search path. For more information, see “Creating a Cross-Project Search Path” on page 113. You need to have at least the *READ project access level to the cross-project in order to be able to export parts from it.

The Part You Want to Export

You must specify the project, group, type, and part name on the EXPPART command. There are special values available for the TYPE and PART parameters. See Appendix D, “Substitution Variables” on page 259 for a list of part types that are allowed with the EXPPART command.

On the TYPE parameter you can enter a specific part type. For example, TYPE(RPGSRC) exports a part of type RPGSRC. You can specify that you want to export parts of all types by using the special value *ALL or you can use a generic name. Use of the generic is described in “Required Parameters” on page 40. You can specify that you want to export all non-source parts by using the special value *NONSRC. If you specify TYPE(*NONSRC), only part types stored in objects, such as those types listed in Table 5 on page 61, are exported. You cannot export a

part whose user-defined type has been defined with the value *NONE on the SYSTYPE parameter of the ADDADMLANG command. Parts of this type already exist in an AS/400 library outside the Application Development Manager/400 project.

On the PART parameter you can enter a specific part name. For example, PART(BIWKLY) exports the part BIWKLY. Specify the special value *ALL to indicate that you want to export all parts or use a generic name. Use of the generic is described in “Required Parameters” on page 40.

Example: This example illustrates how to export all parts of all types from the TEST group in the PAYROLL project to your current library.

Using the EXPPART command

```
EXPPART PRJ(PAYROLL) GRP(TEST) TYPE(*ALL) PART(*ALL)
```

Using the Programming Development Manager utility

Select option 39 (Export) on the Work with Parts Using PDM display.

Example: The following command exports a CSRC part called BIWKLY from the group DEVELOPER1 in the payroll project to your current library.

```
EXPPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(CSRC) PART(BIWKLY)
```

The Object or File You Want to Create

One difference between the IMPPART and EXPPART commands is that on the export operation the *to* criteria have default values. You must indicate on the TOLIB and SRCFILE parameters where you want the exported part to go, or it will be copied to a default library and file. A file is created if one does not already exist in the library.

The default value for the library is the current library. If you do not want the default, indicate a specific library on the TOLIB parameter. For example, TOLIB(TESTLIB) uses the TESTLIB library to store all the copied parts.

The default value for the file name within the library is determined by the name of the source file that contains the actual part in the project hierarchy. To use the default, specify SRCFILE(*FROMFILE).

When you create a part using CRTPART, CPYPART, or IMPPART, you can specify a source file name. If you do not specify a source file name, the part is created in the system-supplied source file that matches the part type. The SRCFILE parameter applies to parts that have one of the part types listed in Table 4 on page 59 or a user-defined type.

Example: The following command creates the part BIWKLY as a source member in the source physical file QRPGSRC.

```
CRTPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY) LANG(RPG)
        SRCFILE(*TYPE)
```

If you specified a source file name when you created the part, when you export that part, it is created in a file of the same name.

Example: The following command creates the part BIWKLY as a source member in the source physical file PAYSRC.

```
CRTPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(MTHLY) LANG(RPG)
        SRCFILE(PAYSRC)
```

When you export the part BIWKLY, it copies the part as a source member in the source physical file PAYSRC in your current library.

```
EXPPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(BIWKLY)
        TOLIB(PAY1) SRCFILE(*FROMFILE)
```

Instead of specifying that you want to export parts into a source file of the same name, you can indicate that you want to use SRCFILE(*TYPE). This value creates the part in the system-supplied default source file for that type, even if the part within the project hierarchy is stored in a file of a different name. See Table 3 on page 49 for a list of default source files. You can also enter a specific source file name on the SRCFILE parameter.

Example: The following command exports the BIWKLY part to the PAY1 library in the file PAYSRC.

```
EXPPART PRJ(PAYROLL) GRP(TEST) TYPE(RPGSRC) PART(BIWKLY) TOLIB(PAY1)
        SRCFILE(PAYSRC)
```

Note that if you export a logical file that already exists in the *To-library* with REPLACE(*YES) specified, and the logical file's access path outside the Application Development Manager/400 environment is journaled, the journaling is reinstated in the *To-library*.

For a description of the SCAN and SCHPTH parameters, see “How Search-Path Parts Are Processed” on page 116. Note that if the part of type SCHPTH identifies a cross-project search path, parts within the other project are *not* exported. You can only export from one project at a time. If you want to export the parts from a shared project, you must enter another EXPPART command and specify that project.

Exporting Data with Parts

The EXPPART command allows you to copy the data associated with your physical data files when exporting your application. The default value for the DATA parameter is *not* to copy data. Specify DATA(*YES) if you want to copy data.

If you specify that you want to copy data on an export operation and the physical data files already exist in the library that you are exporting to, the *files are not replaced*. This is to avoid replacing essential or production data in a production library in error.

If you want to replace a physical data file in a library you are exporting to, manual intervention is required as the EXPPART command does not replace physical data files. You should write a CL program that saves the data from the production copy of the physical file, deletes the file, and then calls the EXPPART command to export the physical file. As a final step, your CL program should restore the saved data to the new physical file.

When you are exporting for testing purposes, you may want to export your test data from the project hierarchy to continue testing. When you are exporting to a production environment, in most cases you do not want to copy the test data.

Example: The following command exports all parts in the PAYROLL project to the library TESTLIB. Data associated with the physical data files is also copied.

```
EXPPART PRJ(PAYROLL) GRP(TEST) TYPE(*ALL) PART(*ALL) SCAN(*YES)
        SCHPTH(*DFT) TOLIB(TESTLIB) SRCFILE(*FROMFILE) DATA(*YES)
        REPLACE(*YES)
```

Authorization Required when Exporting a Part

Because objects are created in a library, certain AS/400 authorization is required to export objects. You must have the necessary authorization to create and delete objects and source file members in the library specified on the TOLIB parameter. If the object or source file member already exists in the library specified, and you specify REPLACE(*YES), you must have the necessary authorization to create and delete that object or source file member. Note that physical data files cannot be replaced.

In addition to the authorization required for the library and objects within that library, the EXPPART command assigns an owner to the objects that are created or replaced. The owner is determined by the value specified on the OWNER parameter. The default value, *TOOBJ, causes the command to use the owner of the object you are replacing. If the object is created, whoever entered the EXPPART command is assigned as the owner.

The ownership of an existing source file does not change when exporting source members. If the EXPPART command automatically creates a new source file, the owner of the source file will be the owner specified on the OWNER parameter. Subsequent exports of members to this source file will not change the ownership of the source file.

If you attempt to change the ownership of an existing object, you must have the authorization to do so.

Usually when you are exporting for testing purposes, you are exporting to a test library that you own. There you should have all the necessary authorization to the library and objects, and should have assigned yourself as the owner.

When you are exporting the part or application to a production environment, it is important that all authorization requirements are met. The export operation may not be able to complete if you do not have the appropriate authorization to all the objects that you need.

Example: The following command exports all the parts in the PAYROLL project to the test library TESTLIB.

```
EXPPART PRJ(PAYROLL) GRP(TEST) TYPE(*ALL) PART(*ALL) SCAN(*YES)
        SCHPTH(*DFT) TOLIB(TESTLIB) SRCFILE(*FROMFILE) DATA(*YES)
        REPLACE(*YES) OWNER(*TOOBJ) AUT(*TOOBJ)
```

Assume this is the first time you have exported this application. You are the owner of the objects created in the TESTLIB library. The public authority for the objects in TESTLIB is determined by the AUT parameter value.

Example: The following command exports all the RPGSRC parts in the group DEVELOPER1 to TESTLIB.

```
EXPPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(RPGSRC) PART(*ALL) SCAN(*YES)
        SCHPTH(*DFT) TOLIB(TESTLIB) SRCFILE(*FROMFILE) DATA(*NO)
        REPLACE(*YES) OWNER(*TOOBJ) AUT(*TOOBJ)
```

To summarize, the EXPPART command does not allow you to do anything to AS/400 objects that you are not authorized to do outside the Application Development Manager/400 environment. For more information on objects and authorities, refer to the AS/400 book, *Security – Reference*.

Exporting a Part-List Part

If you are exporting a part-list part, the EXPPART command allows you to specify how you want to export it. The default PARTLOPT(*LIST) indicates that the parts listed inside the part-list part are what is to be exported: the part-list part itself is not exported. If you specify PARTLOPT(*PART), only the part-list part itself is exported: the parts listed inside the part-list part are not. If you specify PARTLOPT(*BOTH), both the part-list part and the parts listed inside it are exported.

The EXPPART command searches for the part-list part to export in the following order:

1. The search path specified on the EXPPART command is used to search for the part-list part.
2. If PARTLOPT(*LIST) is specified on the EXPPART command, each part is searched for, according to what you specify on the GRP, SCAN, and SCHPTH parameters, and exported. If the part-list part is not found, the EXPPART command fails. If PARTLOPT(*PART) is specified, the part-list part itself is exported in the same way any other part is. If PARTLOPT(*BOTH) is specified, both the part-list part and the parts listed inside it are exported.
3. If PRDDFN and PRDLOD parts are found in the part-list part, the EXPPART command performs some extra processing. See “Exporting and Packaging an Application” on page 189 for information on PRDDFN and PRDLOD parts.

Exporting Using a System-List Part

You can use the EXPPART command to distribute or send your applications, application fixes, or application enhancements to one or more remote AS/400 systems.

A **system-list part** is a part with a part type of SYSTEML that contains a list of system addresses which can be processed to send Application Development Manager/400 parts to one or more remote AS/400 systems. If you specify *NONE on the SYSTEML parameter, the EXPPART command exports the parts to the specified library on the same AS/400 system.

When the system-list part name is specified, the EXPPART command exports parts to the remote AS/400 systems specified in the system-list part.

To find out how to distribute your applications to one or more remote AS/400 system, see Chapter 14, “Distributing Applications to Remote AS/400 Systems” on page 193 for more detailed information.

The sending of distributions using the EXPPART command will fail if there are objects in the QTEMP library when a system-list part has been specified on the SYSTEML parameter.

Exporting and Packaging an Application

The Application Development Manager/400 feature provides two part types to enable you to package your application using the System Manager product. 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.

It is expected that you already understand the steps involved in packaging an application using the SystemView System Manager/400 product. See the *System Manager Use* book.

For packaging to take place during an export operation, a part of type PRDLOD must be found in a part of type PARTL. If a product-load part is found, the parts listed in the part-list part are packaged as the product identified on the Product ID parameter of the Create Product Load (CRTPRDLOD) command that is specified inside the part of type PRDLOD. The CRTPRDLOD command is described in the *System Manager Use* book.

You can have parts of type PRDDFN and type PRDLOD in the part-list part that is being exported. The part of type PRDDFN must contain a valid Create Product Definition (CRTPRDDFN) command. The part of type PRDLOD must refer to the product identified on the Product ID parameter of the Create Product Definition (CRTPRDDFN) command that is specified inside the part of type PRDDFN. The CRTPRDDFN command is also described in the *System Manager Use* book.

Creating a Product Definition and Product Load Part

To create a product-load part, do the following:

1. Use the CRTPART command to create the part. Specify PRDLOD on the TYPE parameter.
2. Use the CHGPART command to add the CRTPRDLOD command and all its parameters and values to the PRDLOD part. You can do this easily by typing the name of the command in the part you are changing and pressing F4=Prompt. The CRTPRDLOD command parameters appear with prefilled default values.

To create a product-definition part, do the following:

1. Use the CRTPART command to create the part. Specify PRDDFN on the TYPE parameter.
2. Use the CHGPART command to add the CRTPRDDFN command and all its parameters and values to the PRDDFN part. You can do this easily by typing the name of the command in the part you are changing and pressing F4=Prompt. The CRTPRDDFN command parameters appear with prefilled default values.

For a description of the CRTPRDLOD and CRTPRDDFN commands, see the *System Manager Use* book. To see help for these commands, enter the command on the command line, press F4=Prompt to prompt the command, and then press F1=Help.

Creating a Part-List Part to Contain the Parts to be Packaged

You must create a part-list part to contain a list of all the parts in your application that need to be included in order for the application to be packaged successfully. See Chapter 8, “Part-List Parts, Reason Control, and Change Tracking” on page 103 for an explanation of how to create and use a part-list part. You must ensure the part-list part contains a part of type PRDLOD. It can also contain a part of type PRDDFN.

Exporting the Part-List Part for Packaging an Application

Use the EXPPART command to export the part-list part you created that contains the product-load part. Specify *LIST on the PARTLOPT parameter of the EXPPART command to export only the parts listed inside the part-list part.

The EXPPART command recognizes that it has exported a list of parts that contains a part of type PRDLOD. It extracts information from the part of type PRDLOD and uses that information to set up the packaging information in all the 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.

Note: If more than one part of type PRDLOD is listed in a part-list part, the EXPPART command uses the first part of this type that it finds. If the part-list part contains nothing but parts of type PRDDFN and PRDLOD, the results of the packaging are unpredictable. You should create a part-list part without the PRDDFN and PRDLOD parts so that the EXPPART command and packaging are successful.

After all the parts have been exported, the EXPPART command runs the CRTPRDDFN command if a part of type PRDDFN that contains the command is found in the part-list part. If the CRTPRDDFN command fails because the *PRDDFN object already exists, the EXPPART command may still be successful.

The CRTPRDDFN command creates an object of type *PRDDFN in the library specified on the Library parameter of the CRTPRDDFN command that was specified inside the product-definition part. You should specify your main product library on the Library parameter of the CRTPRDDFN command.

If the CRTPRDDFN command is not valid, and no product has been defined, no packaging occurs. The EXPPART command next runs the CRTPRDLOD command it finds in the product-load part. If the CRTPRDLOD command fails because the *PRDLOD object already exists, the EXPPART command may still be successful.

The CRTPRDLOD command creates an object of type *PRDLOD in the library specified on the Library parameter of the CRTPRDLOD command that was specified inside the product-load part.

Now the library that you specified on the TOLIB parameter of the EXPPART command contains a copy of all the parts that were listed inside the part-list part.

Setting Up the Packaging Information

The EXPPART command sets up the packaging information in the following manner:

1. The Change Product Object Description (CHGPRDOBJD) command is run against each object that was exported, as listed in the part-list part, to update the product information in the object description. If you are exporting parts that are stored in source file members, the CHGPRDOBJD command is run against the source file.
2. The Package Product Option (PKGPRDOPT) command is run automatically using the information specified in the product load part. This command packages one or more product loads for a specified product option enabling the loads to be saved using the Save Licensed Program (SAVLICPGM) command.

At this point, you use the SAVLICPGM command to save a copy of all of the objects that make up a licensed program. This command saves the licensed program in a form that can be restored by the Restore Licensed Program (RSTLICPGM) command. Before you use the SAVLICPGM command, you should verify that all the objects that make up your product are listed in the product library. The *System Manager Use* book describes the conditions that must be met for the packaging step to be successful. You should ensure these conditions are met before you run the SAVLICPGM command.

If your product consists of more than one AS/400 library, you might need to use the EXPPART command more than once. You will have to create a part-list part for each additional EXPPART command you run. For each part-list part, you must also create another product-load part. The *System Manager Use* book provides CL command examples of the CRTPRDLOD command in the case where a product can exist in more than one library.

Chapter 14. Distributing Applications to Remote AS/400 Systems

A **system-list part** is a part with a part type of SYSTEML that contains a list of system addresses which can be processed to send Application Development Manager/400 distributions to one or more remote AS/400 systems. **Distributions** are Application Development Manager/400 parts which are exported to a remote AS/400 system. A system-list part can be used to distribute applications, application fixes, and application enhancements to remote systems.

This chapter provides information on the following topics:

- Distributing receive programs
- Building previous release applications
- Creating a system-list part
- Distributing applications to remote AS/400 systems
- Receiving objects from a remote AS/400 system

Distributing Receive Programs

In order to receive distributions using the Receive Part (RCVPART) command from remote AS/400 systems, a subset of the Application Development Manager/400 programs that support this command must first be installed on the remote systems. The remote AS/400 systems must also be at V3R7, or a previous release (for example, V3R6).

If the remote AS/400 systems are at V3R7 and have the Application Development Manager/400 feature installed, skip this section.

The procedure for preparing remote systems (without Application Development Manager/400 installed) to receive distributions is described in the following sections.

The instructions vary slightly depending on whether you are dealing with Reduced Instruction Set Computers (RISC) or Internal Machine Product Instruction (IMPI) systems. Programs can be received from a current release system by both RISC (V3R6, V3R7) and IMPI (V3R1, V3R2) systems.

V3R7 System without Application Development Manager/400: To distribute the receive programs that support the Receive Part (RCVPART) command to a V3R7 AS/400 that does **not** have the Application Development Manager/400 feature installed, you must do the following:

1. At the sending system (a system with the V3R7 Application Development Manager/400 feature already installed), create a savefile QADMDIST in the QADM library by typing:

   ```
   CALL QADM/QLYSAVDST
   ```

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

   The QLYSAVDST program will then complete the following tasks:

   a. Creates the QADMDIST library

b. Copies the list of objects specified in Table 16 on page 194 from the QADM into the QADMDIST library. Table 16 on page 194 also indicates whether the objects are Machine Readable Information (MRI) or Machine Readable Material (MRM).

Table 16. MRI and MRM Objects Copied from QADM into QADMDIST Library for RISC System

Objects	Object Type	Machine Readable Type
QADMMSG	*MSGF	MRI
QHLYCMD	*PNLGRP	
RCVPART	*CMD	
QLYCHKQT	*PGM	MRM
QLYCPP		
QLYCPYAA		
QLYDSFIL		
QLYDSTYP		
QLYEXEAA		
QLYIEXIT		
QLYINIT		
QLYMOVAA		
QLYRCVP		
QLYVRFY		

c. Creates a savefile called QADMDIST in the QADM library

d. Saves the QADMDIST library into the savefile using the *CURRENT value on the *Target release* parameter of the Save Library (SAVLIB) command.

e. Deletes the QADMDIST library

Note: Since all these objects are copied from the QADM library, the machine readable information (MRI) objects will be in the primary language of the local system in which the savefile is created. If you want to create a savefile that contains the MRI objects in any other language, then on the local system you must first install the Application Development Manager/400 feature in that language as the secondary language and perform the above steps manually. Copy the Receive Part (RCVPART) command, the message file, and the panel group objects from the secondary language library QSYS29xx, where xx is the language identifier. Then copy all the program objects listed in Table 16 (or machine readable material, MRM) from the QADM library into the QADMDIST library. Then perform the above steps 1c to 1e.

2. Send the savefile to each remote RISC system where your users want to use the Receive Part (RCVPART) command.
3. At the receiving system, receive the savefile.
4. On the receiving system, restore the savefile into the QADMDIST library, using the Restore Library (RSTLIB) command.

Previous IMPI System with or without Application Development Manager/400:
To distribute the needed programs that support the Receive Part (RCVPART) command to a previous release of an IMPI AS/400 system (for example, V3R2) with or without the Application Development Manager/400 feature installed, you must do the following:

1. At the sending system (a system at the current release) with the Application Development Manager/400 feature installed), create the two savefiles, QADMDISTP and QADMDISTP2, in the QADM library by typing:

   ```
   CALL QADM/QLYSAVDSTP
   ```

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

   The QLYSAVDSTP program will then complete the following tasks:

   a. Creates two savefiles called QADMDISTP and QADMDISTP2 in the QADM library

   b. Saves the list of objects specified in Table 17 from the QADMDISTP library into the savefile QADMDISTP using the V3R6M0 and into the savefile QADMDISTP2 using the V3R1M0 value on the *Target release* parameter of the Save Object (SAVOBJ) command

Table 17. MRI and MRM Objects Copied from QADM into QADMDISTP Library for IMPI System

Objects	Object Type	Machine Readable Type
QADMMSG	*MSGF	MRI
QHLYCMD	*PNLGRP	
RCVPART	*CMD	

Note: Since some MRI objects are copied from the QADM library, they will be in the primary language of the local system in which the savefile is created. If you want to create a savefile that contains the MRI objects in another language, then on the local system you must first install the Application Development Manager/400 feature in that language as the secondary language, and perform the above steps manually.

Do not replace the Receive Part (RCVPART) command in the QADMDISTP library. Copy the message file and the panel group objects from the secondary language library QSYS29xx, where xx is the language identifier, into the QADMDISTP library. Then, complete steps 1a and 1b. You may want to delete the message file and the panel group objects from the QADMDISTP library, after the savefiles are created. Even though the Receive Part (RCVPART) command will be saved in the primary language of the local system, it should not have any significant impact, as it does not contain any parameters. The online help and the messages issued by the Receive Part (RCVPART) command on the remote system will be in the secondary language.

You may want to save a copy of the primary language version of QADMDISTP and QADMDISTP2, and then restore them after you create the QADMDISTP and QADMDISTP2 savefiles.

2. Send the QADMDISTP2 savefile from the QADM library to each remote IMPI system from which your users want to use the Receive Part (RCVPART) command.
3. Send the QADMPGM2 savefile from the QADMDISTP library to each remote IMPI system from which your users want to use the Receive Part (RCVPART) command.
4. At the receiving system, receive these savefiles.
5. On the receiving system, create the QADMDISTP library.
6. Restore the savefile QADMDISTP2 into the QADMDISTP library, using the Restore Object (RSTOBJ) command.
7. Restore the savefile QADMPGM2 into the QADMDISTP library, using the Restore Object (RSTOBJ) command.

Note: In order to receive the distributions properly, you also need to install Program Temporary Fixes (PTF) MF08977 and MF09416 for the Licensed Program Product (LPP) 5763999 on the receiving V3R1M0 IMPI systems.

Previous RISC System with or without Application Development Manager/400: To distribute the needed programs that support the Receive Part (RCVPART) command to a previous release of a RISC AS/400 system (for example, V3R6) with or without the Application Development Manager/400 feature installed, you must do the following:

1. At the sending system (a system at the current release with the Application Development Manager/400 feature installed), create a savefile QADMDISTP in the QADM library by typing:

   ```
   CALL QADM/QLYSAVDSTP
   ```

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

   The QLYSAVDSTP program will then complete the following tasks:

   a. Creates two savefiles called QADMDISTP and QADMDISTP2 in the QADM library
   b. Saves the list of objects specified in Table 18 from the QADMDISTP library into the savefile QADMDISTP using the V3R6M0 and into the savefile QADMDISTP2 using the V3R1M0 value on the *Target release* parameter of the Save Object (SAVOBJ) command

Table 18. MRI and MRM Objects Copied from QADM into QADMDISTP Library for RISC System

Objects	Object Type	Machine Readable Type
QADMMSG	*MSGF	MRI
QHLYCMD	*PNLGRP	
RCVPART	*CMD	

Note: Since some MRI objects are copied from the QADM library, they will be in the primary language of the local system in which the savefile is created. If you want to create a savefile that contains the MRI objects in another language, then on the local system you must first install the

Application Development Manager/400 feature in that language as the secondary language, and perform the above steps manually.

Do not replace the Receive Part (RCVPART) command in the QADMDISTP library. Copy the message file and the panel group objects from the secondary language library QSYS29xx, where xx is the language identifier, into the QADMDISTP library. Then, complete steps 1a on page 196 and 1b on page 196. You may want to delete the message file and the panel group objects from the QADMDISTP library, after the savefiles are created. Even though the Receive Part (RCVPART) command will be saved in the primary language of the local system, it should not have any significant impact, as it does not contain any parameters. The online help and the messages issued by the Receive Part (RCVPART) command on the remote system will be in the secondary language.

You may want to save a copy of the primary language version of QADMDISTP and QADMDISTP2, and then restore them after you create the QADMDISTP and QADMDISTP2 savefiles.

2. Send the QADMDISTP savefile from the QADM library to each remote RISC system from which your users want to use the Receive Part (RCVPART) command.
3. Send the QADMPGM savefile from the QADMDISTP library to each remote RISC system from which your users want to use the Receive Part (RCVPART) command.
4. At the receiving system, receive these savefiles.
5. On the receiving system, create the QADMDISTP library.
6. Restore the savefile QADMDISTP into the QADMDISTP library, using the Restore Object (RSTOBJ) command.
7. Restore the savefile QADMPGM into the QADMDISTP library, using the Restore Object (RSTOBJ) command.

Building Previous Release Applications

If you want to distribute your applications that run on the AS/400 systems at previous releases supported by the compilers, you must first build the applications using the appropriate value on the *Target release* parameter of the compiler commands.

For each target release identified:

1. Create a separate group such that the group that contains your production-version code is its parent group.
2. Create a build options part named QDFT in each such group with the promote code *NONE.
3. Uncomment all the compiler commands required.
4. Add the appropriate release values in the *Target release* parameter.
5. Build entire applications in this group using previous-release compilers.

If you are using any other BLDOPT parts with specific names, or BLDOPT *compiler-names*, then you must repeat the above steps for each individual part.

For more information about the build options parts, see “Using Build Options” on page 157.

Creating a System-List Part

A system-list part can be used to distribute applications, application fixes, and application enhancements to one or more remote systems. To create a system-list part, use the Create Part (CRTPART) command.

Example: The following command creates a system-list part in the group DEVELOPER1 in the project PAYROLL. The part will be called TESTSYS.

```
CRTPART PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(SYSTEML) PART(TESTSYS)
        TEXT('Test systems used for product A')
```

The default source file used for system-list parts is QSYSTEMSRC.

Use the Change Part (CHGPART) command to add remote system addresses to the system-list part. For example, for the above system-list part you can add the following entries:

```
TESTAS01        AS/400 Test machine 001
TESTAS02        AS/400 Test machine 002
TESTAS03        AS/400 Test machine 003
```

You can send distributions to any remote AS/400 system even if they do not have receive capability. If the remote system does not have receive capability then this means that users must receive these distributions manually.

The first eight characters of each record in a SYSTEML part must be a system address defined in the local system directory. The rest of each record can be filled with any information that you find useful. In the above example, a brief description of each machine is given.

It is possible that each Application Development Manager/400 project might have multiple system-list parts created. For example, there could be one system-list part for each release of the application being stored in the project. There could be one system-list part for the current release containing the list of systems on the current release, and another one for the previous release. Individual developers may also want to set up a system-list part that contains only the name of the system to which they want to distribute their fixes.

Distributing Applications to Remote AS/400 Systems

Specifying the name of a system-list part on the SYSTEML parameter of the Export Part (EXPPART) command causes it to export parts to all the systems listed in the system-list part. Your user profile is determined and the corresponding user ID from the system directory is obtained. The distributions are sent to that user ID on each system address listed in the system-list part.

Specifying a SYSTEML name on the SYSTEML parameter of the Export Part (EXPPART) command exports the distributions to all the remote AS/400 systems listed in the SYSTEML part. Parts are not exported to the local AS/400 system. The

parts must exist in the default search path starting from the specified group up to the root group.

The TOLIB, SRCFILE, DATA, REPLACE, OWNER, and AUT parameters specified on the Export Part (EXPPART) command are not processed on the local system. Instead, these values are sent along with the distribution to each remote system address listed in the system-list part and processed by the Receive Part (RCVPART) command there.

The Export Part (EXPPART) command will fail if you are not enrolled in the local system's directory. The distributions will fail if the user IDs you are sending to are not enrolled on the remote system's directory. The user ID can be enrolled on a system using the Add Directory Entry (ADDDIRE) command.

The TGTRLS parameter is where you specify the release of the operating system to which distributions are to be sent. This parameter is ignored if SYSTEML(*NONE) is specified. You can use the default *CURRENT, *PRV, or any of the choices listed either in the contextual help or on the prompt display for the TGTRLS parameter.

Receiving Objects from a Remote AS/400 System (RCVPART)

Use the Receive Part (RCVPART) command to receive the incoming Application Development Manager/400 distributions from one or more remote AS/400 systems. **Incoming distributions** are parts which are sent from a remote AS/400 system using the Export Part (EXPPART) command. The Receive Part (RCVPART) command processes only those distributions sent by a user with a user ID that is identical to that of the current user.

There are no parameters on the Receive Part (RCVPART) command.

For each distribution, the Receive Part (RCVPART) command moves the incoming parts into the library that was specified on the Export Part (EXPPART) command issued on the remote system. In addition, the SRCFILE, DATA, REPLACE, OWNER, and AUT parameters specified on the Export Part (EXPPART) command are processed exactly as if the parts were being exported from the local system.

If you want to receive the distributions at a regular interval to receive fixes, you might want to set up a batch job to run the Receive Part (RCVPART) command at certain times.

| ***Current Release System with Application Development Manager/400:*** To receive incoming Application Development Manager/400 distributions from one or more remote AS/400 systems with the current release of Operating System/400 installed, use the Receive Part (RCVPART) command.

| ***Current Release System without Application Development Manager/400:*** To receive incoming Application Development Manager/400 distributions from one or more remote AS/400 systems with the current release of Operating System/400 installed:

1. Add QADMDIST to the user portion of the library list by typing `ADDLIBLE QADMDIST` on any AS/400 command line.
2. Run the Receive Part (RCVPART) command.

Previous Release System without Application Development Manager/400: To receive incoming Application Development Manager/400 distributions from one or more remote AS/400 systems supporting Application Development Manager/400 distributions:

1. Add QADMDISTP to the user portion of the library list by typing `ADDLIBLE QADMDISTP` on any AS/400 command line.
2. Run the Receive Part (RCVPART) command.

Previous Release System with Application Development Manager/400: To receive incoming Application Development Manager/400 distributions from one or more remote AS/400 systems supporting Application Development Manager/400 distributions:

1. Exit PDM, if you are using it.
2. Add QADMDISTP to the system portion of the library list by typing `CHGSYSLIBL QADMDISTP` on any AS/400 command line.
3. Reclaim resources using the Reclaim Resources (RCLRSC) command.
4. Run the Receive Part (RCVPART) command.
5. Remove QADMDISTP from the system portion of the library list by typing `CHGSYSLIBL QADMDISTP REMOVE(*YES)` on any AS/400 command line.
6. Reclaim resources using the Reclaim Resources (RCLRSC) command.

Any Other AS/400 System: On a system that does not have the Receive Part (RCVPART) command, you may manually receive incoming Application Development Manager/400 distributions from one or more remote AS/400 systems.

To receive the incoming Application Development Manager/400 distributions manually:

1. Sign on with the same user ID as the sender on the receiving AS/400 system.
2. On any AS/400 command line, type `WRKNETF`. Incoming distributions will have a name of QADMSAVF.
3. Receive these save files into a library.
4. Restore all the objects in the save files using the Restore Object (RSTOBJ) command.

   Note that each distribution will contain two extra data areas (QADMDATA and QADMTOLIB) which are not part of the distribution and should be deleted.

Restrictions

1. Because the distribution algorithm uses library QTEMP, it is not possible to send parts to library QTEMP at remote systems. That is, TOLIB(QTEMP) will not be allowed when a system-list part has been specified on the SYSTEML parameter.
2. It is not possible to send or receive a distribution when there are already objects in the QTEMP library. The Receive Part (RCVPART) or Export Part (EXPPART) commands will fail. Clear the QTEMP library and reissue the command.
3. You must be enrolled in the system directory in order to send or receive Application Development Manager/400 distributions.

Chapter 15. Securing Application Development Manager/400 Information

This chapter describes:

- Security in the Application Development Manager/400 feature
  - Additional project security considerations
- Backup and recovery strategies
  - System disaster recovery
  - Abnormal end to Application Development Manager/400 command processing
  - Moving project information to another AS/400 system
- Using the project log
- Maintaining the project hierarchy when a user leaves
- Considerations when an administrator works as a developer

Note

Because the security the Application Development Manager/400 feature provides is based on the security the OS/400 system provides, it is recommended that the Application Development Manager/400 feature be installed on a system with a security level of 30 or higher.

Security in the Application Development Manager/400 Feature

When this feature is installed, a user profile called QPRJOWN is created with *USER level authority and no password. This user profile is the owner of all the objects and libraries that the Application Development Manager/400 feature creates to hold parts.

Note: Be aware that unpredictable results may occur if you change the object authority of the QPRJOWN object of type *USRPRF in library QSYS. Usually, objects that are created when a user runs one of the Application Development Manager/400 commands are owned by this profile. Public authorizations are set to *EXCLUDE, and all private authorizations are revoked. For example, when the CRTGRP command is run to create a group and its associated library, that library is owned by the user profile QPRJOWN, and no user is given private authorizations to the library. As a result, users are very restricted if they attempt to adopt owner authority on an object under Application Development Manager/400 control.

Access by administrators and developers to the objects owned by QPRJOWN is controlled using *authorization lists*. For each group created in a project hierarchy, two authorization lists are used: one to secure the AS/400 library that corresponds to the group, and the other to secure the AS/400 objects that correspond to the parts in that group. The name of each authorization list is a combination of the short project and short group names.

Each of these authorization lists initially contains only one entry, specifying *EXCLUDE authority rights for *PUBLIC. As each user is enrolled to a project, an entry is added to each authorization list with the correct authority for that user.

A user's authority to access parts depends on how he or she is enrolled in a project. Administrators have *ALL authority rights to all groups in a project, while developers have *USE authority rights to all nondevelopment groups. Developers are only authorized to do part development in those development groups to which they have *UPDATE access.

Users who are members of a group profile generally give other users in that group additional authorization to objects that they create. The Application Development Manager/400 feature does not allow this to occur.

In addition, it is customary for AS/400 system administrators to enroll users in a group profile to manage the authorizations of those users as a group. Although the Application Development Manager/400 feature allows you to enroll a group profile to a project, it does not honor the members within that profile. Therefore, it is not recommended that group profiles be enrolled into a project. Users must be enrolled explicitly into a project to perform Application Development Manager/400 commands against the parts in the project.

Additional Project Security Considerations

The Application Development Manager/400 feature introduces several methods of controlling the application development process. The action of checking out a part ensures that only one developer has access to the part at a time. This control is enforced when Application Development Manager/400 interfaces are used to perform actions on Application Development Manager/400 information. You should use the Application Development Manager/400 CL commands or the Programming Development Manager utility to perform all activities.

If you decide to change a part using a non-Application Development Manager/400 interface, unpredictable results can occur. For example, the BLDPART command may not recognize that a part has changed, and thus not re-build it. This can lead to inconsistencies between source parts and output parts.

In addition, you should not create objects or files in Application Development Manager/400 project libraries. All information within project libraries should be created through the Application Development Manager/400 feature as individual parts. If you want to copy files or objects from one library into the project hierarchy, use the IMPPART command.

You should not place objects or files into a project library without first creating those objects or files using Application Development Manager/400 commands. Otherwise, these objects or files are not recognized as Application Development Manager/400 information.

Backing Up and Recovering Application Development Manager/400 Information

A system administrator should back up the entire system regularly to ensure that the organization can recover in the event of a system failure. Application Development Manager/400 information needs to be backed up as part of your organization's regular backup and recovery procedures.

In addition to performing regular system backup and, if necessary, recovery, sometimes individual commands do not finish processing and need to be recovered. Command processing can be interrupted because a system administrator canceled a job; a power failure occurred; or a user selected option 2 from the system request menu to end the previous request. For some Application Development Manager/400 commands, information can be left in an **inconsistent state** if a command does not finish its processing normally. See “Recovering from a Command Processing Failure” on page 206 for information on how to recover from situations like this.

Backup and Recovery Strategies

The AS/400 backup and recovery strategy as described in *Backup and Recovery – Basic*, should be used to back up and recover Application Development Manager/400 information along with all your other system information. If your AS/400 system is not backed up and recovered on a regular basis, or if only certain libraries are saved, any attempt to restore Application Development Manager/400 information will be unpredictable. Use one of the strategies described in *Part 1. Developing a Backup and Recovery Strategy* in *Security – Basic*. If you do not use one of the backup and recovery strategies suggested there, unpredictable results may occur when you use the Application Development Manager/400 feature.

If the user profiles are restored or recreated as part of the recovery process, you may receive the message ADM1804 when you try to use any of the Application Development Manager/400 commands. To correct this situation, sign on as QSECOFR. Remove all project users using the RMVPRJUSR command and then add the users back to their respective projects using the ADDPRJUSR command.

Besides following regular AS/400 system backup and recovery procedures, the project's administrator must use the Reclaim Project (RCLPRJ) command to ensure that all Application Development Manager/400 information is restored correctly after libraries and objects are restored in the event of a system failure. This additional step is required because the information restored by the AS/400 system commands does not include information about objects that have been deleted or moved. In addition, Application Development Manager/400 journal files are not saved, but their journal receivers are. This can create inconsistencies in the parts in project hierarchies and in the information about these parts.

When you recover information within a project hierarchy, the location of parts within the project hierarchy is part of the information you require. For example, assume the part PAYROLL DEVELOPER1 RPGSRC BIWKLY is promoted out of its development group and into a test group. In the event of a system failure requiring a restore of system data, you will want the part to be removed from the group DEVELOPER1 and restored in the group TEST when you restore your Application Development Manager/400 information.

When you periodically remove or delete parts from certain groups—for example, when you build the part PAYROLL DEVELOPER1 RPGSRC BIWKLY—the part created (PAYROLL DEVELOPER1 PGM BIWKLY) remains in the development group until it is deleted. Once the part is deleted, you do not want it to reappear after you restore the system.

Using the RCLPRJ command resolves any inconsistencies present in Application Development Manager/400 parts after restoring the libraries and objects as discussed earlier. The project hierarchy is restored to its original, or correct, state because in addition to saving AS/400 files and objects, the AS/400 save and restore commands also save and restore the information that the Application Development Manager/400 feature creates and maintains about those files and objects. This is why it is important to use the recommended AS/400 save and restore strategy instead of saving and restoring only particular libraries.

Reclaiming a Project (RCLPRJ)

You must be a project administrator to reclaim projects. An administrator is required to run the Reclaim Project (RCLPRJ) command before doing anything else after a system has been restored. The first time this command is run on an AS/400 system after a journal restore, all journal changes are applied to the Application Development Manager/400 databases for all projects. However, each project must still be reclaimed by its project administrator. If anyone attempts to use a project that needs to be reclaimed, an error message is issued advising him or her that the project needs to be reclaimed.

The RCLPRJ command uses the information restored by the system backup and recovery procedures to ensure that the parts within the project hierarchy are put back in the state they were found when the information was saved. In other words, parts that were promoted or deleted are accurately represented in the project hierarchy after the RCLPRJ command is processed.

The information the Application Development Manager/400 feature creates and maintains about files and objects is called **part directory information**. Part directory information contains information on when the part was created, when it was last changed, and where it resides in the project hierarchy. This information is compared with the actual files and objects that are restored when the RCLPRJ command is run. If inconsistencies exist, the Application Development Manager/400 feature resolves them.

For example, assume the part directory information indicates that a part no longer exists in the development group. However, the actual object was restored in the library representing the development group. The part should be deleted. The RCLPRJ command allows you to delete all files or objects within the project hierarchy that do not have corresponding part directory information by using the default *RUN on the OPTION parameter and *YES on the DLTINCOBJ parameter. Only objects that have system-supplied part types or objects that have user-defined part types are deleted.

Note

If you have created files or objects within Application Development Manager/400 project libraries and did not use the Application Development Manager/400 feature to do so, these files or objects are deleted when you run the RCLPRJ command with OPTION(*RUN) and DLTINCOBJ(*YES) if they are of a type supported by the Application Development Manager/400 feature. For a list of supported part types, see Table 4 on page 59 and Table 5 on page 61. Any objects that do not have corresponding Application Development Manager/400 types (either system-supplied types or user-defined types) are neither reported nor deleted.

You must specify the name of the project on the RCLPRJ command, or indicate that you want to reclaim project information for all projects to which you are authorized as an administrator.

The DLTINCOBJ parameter is where you specify if you want **inconsistent objects** deleted. Inconsistent objects are objects for which associated part directory entries are not found.

Example: This example illustrates how to reclaim information for the sample payroll project and deletes all the objects and members within this project that do not have associated part directory information.

Using the RCLPRJ command

```
RCLPRJ PRJ(PAYROLL) OPTION(*RUN)
```

Using the Programming Development Manager utility

Select option 37 (Reclaim) on the Work with Projects Using PDM display.

Example: When you run the RCLPRJ command, a report is produced that indicates which objects or files do not have the necessary part directory information. To print a RCLPRJ report without deleting any parts, enter the following command.

```
RCLPRJ PRJ(PAYROLL) OPTION(*TEST)
```

Figure 61 illustrates the information provided in the Reclaim Project report. In this example, five parts are not recognized according to the part directory information.

```
5716PW1   V3R7M0              Application Development Manager/400 - Reclaim Project 11/08/96    10:35:57         Page . . :  0001

Project . . . . . . . . . . . . . . . :   PAYROLL
Option  . . . . . . . . . . . . . . . :   *RUN
Delete inconsistent objects . . . . . :   *NO
AS/400                 Object      Source      Member      Create
Library     Object     Type        Member      Type        Date     Owner       Text

PAY.PROD    ASSISTS    PGM                                 11/08/96 RICHARDS
PAY.PROD    GOALS      PGM                                 11/08/96 RICHARDS
PAY.PROD    PENALTIES  PGM                                 11/08/96 RICHARDS
PAY.PROD    POINTS     PGM                                 11/08/96 RICHARDS
PAY.TEST    QDDSSRC    FILE        NEWSTATS    PF          11/08/96 QPRJOWN
            * * * * * E N D  O F  L I S T I N G  * * * * *
```

Figure 61. A Sample Report Produced by the RCLPRJ Command

If you indicate that you want to reclaim information about all projects to which you are authorized, a separate report is produced for each project.

Objects found in Application Development Manager/400 project libraries that do not have system-supplied or user-defined types defined for them are neither reported nor deleted by the RCLPRJ command.

Recovering from a Command Processing Failure

When Application Development Manager/400 commands that act on both the part directory information and the object or file are interrupted for some reason before both steps are complete, project information is left in an inconsistent state. This could be caused by a system operator or security officer canceling a job before processing is finished by a user ending a previous request through option 2 from the system request menu, or by a power failure.

Project information is not left in an inconsistent state because a command ends normally and provides an error message to the user. For example, if the CRTPRJ command returns a message indicating that the command failed because the short project name is not valid, all project information is still consistent.

Part development commands are handled slightly differently from project administration commands when project information is found to be inconsistent.

Recovering Part Development Commands

When part development commands fail to process completely, manual intervention is usually not required to recover parts after the inconsistency is discovered. The inconsistency is not discovered until a developer or an administrator attempts to use or change an inconsistent part, at which time the Application Development Manager/400 feature automatically resolves the error situation.

Part development commands do allow for part directory information to exist even if the corresponding file or object does not. This means that if the CRTPART command fails, the part directory information could exist although the part itself does not. When part directory information exists for a part (a file or object) that does not exist, an administrator or developer can *see* this part on a QRYPART list.

If a user attempts to perform some action with this part, the Application Development Manager/400 feature determines that only the part directory information exists, and not the part itself. The part directory information is deleted and the command does not find the part in that group. If a later QRYPART command is entered, the part is no longer listed.

If the part that the command was acting on is damaged and no longer usable, the backup copy of the object or file needs to be restored using AS/400 backup and recovery commands. See “Backup and Recovery Strategies” on page 203 for information on how to do this.

Recovering Project Administration Commands

Project administration commands are recovered manually either by entering the command again or by entering the RCLPRJ command.

If the CRTPRJ, CHGPRJ, or DLTPRJ command fails, the administrator should enter the same command again. For example, if the CRTPRJ command is entered but fails because of a power outage, anyone attempting to access this project, be they administrator or developer, sees a message saying that the project does not exist. The administrator should enter the same CRTPRJ command again.

If one of the following commands fails when processing, the RCLPRJ command must be run with OPTION(*RUN) specified to return the project information to a consistent state.

ADDPRJUSR	CRTGRP	RMVPRJUSR
CHGPRJUSR	DLTGRP	

Copying Information to Another AS/400 System

If you want to move or copy a project hierarchy and all the information associated with it to another AS/400 system, use the EXPPART and IMPPART commands. You can also move your Application Development Manager/400 project libraries and archive libraries to separate auxiliary storage pools (ASP), as long as the entire project resides in the same ASP. See Chapter 13, “Exporting an Application” and Chapter 6, “Importing an Application” for information on these commands.

Note that the project hierarchy structure itself is not copied. The administrator must create the project and groups within the project using the appropriate Application Development Manager/400 commands. See “Creating the Project Hierarchy” on page 17 for information on how to do this.

The access or enrollment information must also be created for the new project hierarchy. See “Enrolling Users in a Project Hierarchy” on page 31 for information on how to do this.

There are the following two ways you can achieve this: single version application and multiple versions application.

Single Version Application

If your applications do not have multiple versions of the code, or the code is not scattered over multiple groups, then this method might be better for you.

All the parts within the project hierarchy you want to copy should be in the root group. Then you can use the EXPPART command to export all the parts from this group into a library. This library can be saved and restored on the other AS/400 system. The library that contains all the Application Development Manager/400 information is used with the IMPPART command on the second AS/400 system.

After the new project hierarchy has been created and the user enrollment information has been added, you should import all parts into one group and test them within that group to ensure that the application works as expected.

Multiple Versions Application

You can use this method to move or copy a project hierarchy which contains the code scattered over multiple groups, or the application which has multiple versions. You can use the EXPPART command with SCAN(*NO) for each group of the project to export all the parts from this group into a library. You need to repeat this step for each group of the project, creating separate libraries for each group. Now you can save all the libraries you created and restore them on the other AS/400 system.

After the new project hierarchy has been created and the user enrollment information has been added on the other system, you should import all parts using REPLACE(*YES) using the IMPPART command from the restored library into its corresponding group. You need to repeat this step for each group starting from the root group to the node group in any branch of the project hierarchy.

Note that this method will be useful even for importing a brand new application with multiple versions to the Application Development Manager/400 environment.

Using the Project Log

To provide an audit trail of activity against projects, the Application Development Manager/400 feature maintains a **project log**. The project log keeps track of the actions performed by developers and administrators on a project by recording what parts are created or changed, when the activity occurs, and by whom. All the Application Development Manager/400 projects on a given AS/400 system are recorded in this project log.

The project log is implemented as an AS/400 journal. This journal and its journal receivers reside in the QUSRSYS library. The journal receivers for the project log must reside here to be processed by the Print Project Log (PRTPRJLOG) command. No user, not even someone with QSECOFR authority, can delete or change individual journal entries. This ensures the security of the project log.

The name of the journal containing the project log is QLYPRJLOG. The name of the journal receivers is QLYPRJxxxx, where xxxx is a number from 1 to 9999. Journal entries are stored in the user-defined entry format.

This product automatically monitors the journal receivers and attaches new journal receivers when existing ones are approaching the threshold limit. Information messages are sent to the QSYSOPR message queue notifying the operator when this is done. You can save the old journal receiver and subsequently delete it to free storage. The recommended strategy is to do this once a week. For more information on journaling, see *Backup and Recovery – Advanced*.

Printing Project Log Information (PRTPRJLOG)

Use the PRTPRJLOG command to see the contents of the project log. It is recommended that this command be run in batch, depending on the extent of information to be printed in the project log. Table 19 on page 209 lists the commands that are logged.

Table 19. Commands Logged in the Print Project Log

Type of Commands	Command Name
Project Administration	ADDPRJUSR, CHGGRP, CHGPRJ, CHGPRJUSR, CRTGRP, CRTPRJ, DLTGRP, DLTPRJ, RCLPRJ RMVPRJUSR
Part Development	ADDADMLANG, ADDADMTYPE, BLDPART, CHGADMACN, CHGADMLANG, CHGPART, CHGPARTINF, CHKINPART, CHKOUTPART, CPYPART, CRTPART, CVTPART, DLTPART, EXPPART, IMPPART, MRGPART, PRMPART, RMVADMLANG, RMVADMTYPE

Some part development commands allow many parts to be acted on with a single command. For example, the PRMPART command can be used to promote all parts in a group. Except for the BLDPART command, a log entry is made for each part acted on by these commands. For BLDPART, only one entry is made, and it shows the BLDPART command as it was entered by the user.

Example: The only required parameter on the PRTPRJLOG command is the name of the project for which you want to see entries. To print a report that describes today's activities for all the developers and administrators using Application Development Manager/400 commands on the payroll project, use the following command. Figure 62 illustrates the report that is created.

```
PRTPRJLOG PRJ(PAYROLL)
```

```
5716PW1   V3R7M0            Application Development Manager/400 - Print Project Log  11/08/96    10:29:07         Page . . :  0001

Project . . . . . . . . . . . . . . . :   PAYROLL
Beginning Date  . . . . . . . . . . . :   11/08/96
Ending Date . . . . . . . . . . . . . :   11/08/96
Search Strings  . . . . . . . . . . . :

User Profile Date       Time       Command

TREMBLAY     11/08/96   10:07:52   CRTPRJ PRJ(PAYROLL) SHORTPRJ(PAY) SAVDTA(*YES) TEXT('*BLANK')
TREMBLAY     11/08/96   10:08:42   CRTGRP PRJ(PAYROLL) GRP(PRODUCTION) SHORTGRP(PROD) PARENT(*NONE) PRMCODE(*PARENT) CCSID(*P
                                   ARENT) TEXT('*BLANK')
TREMBLAY     12/22/95   10:09:10   CRTGRP PRJ(PAYROLL) GRP(TEST) SHORTGRP(TEST) PARENT(PRODUCTION) PRMCODE(*PARENT) CCSID(*PA
                                   RENT) TEXT('*BLANK')
TREMBLAY     12/22/95   10:09:42   CRTGRP PRJ(PAYROLL) GRP(SMITH) SHORTGRP(SMITH) PARENT(TEST) PRMCODE(*PARENT) CCSID(*PARENT
                                   ) TEXT('*BLANK')
TREMBLAY     12/22/95   10:10:21   CRTGRP PRJ(PAYROLL) GRP(DOUG) SHORTGRP(DOUG) PARENT(TEST) PRMCODE(*PARENT) CCSID(*PARENT)
                                   TEXT('*BLANK')
TREMBLAY     12/22/95   10:11:38   ADDPRJUSR PRJ(PAYROLL) USRPRF(SMITH) USRTYPE(*DEVELOPER) ACCESS(*UPDATE) DEVELOPGRP(TANYA)
TREMBLAY     12/22/95   10:12:10   ADDPRJUSR PRJ(PAYROLL) USRPRF(DOUG) USRTYPE(*DEVELOPER) ACCESS(*UPDATE) DEVELOPGRP(DOUG)
SMITH        12/22/95   10:13:45   CRTPART PRJ(PAYROLL) GRP(SMITH) TYPE(DDSSRC) PART(MTHLY) LANG(PF) PROMPT(*NO) PRMCODE(*GRP
                                   ) SRCFILE(*TYPE) TEXT('*BLANK')
SMITH        12/22/95   10:14:07   CHKOUTPART PRJ(PAYROLL) GRP(SMITH) TYPE(DDSSRC) PART(MTHLY) PRMCODE(*GRP)
SMITH        12/22/95   10:15:06   CHGPART PRJ(PAYROLL) GRP(SMITH) TYPE(DDSSRC) PART(MTHLY) CHGCMD(*TYPE)
SMITH        12/22/95   10:15:25   BLDPART PRJ(PAYROLL) GRP(SMITH) TYPE(DDSSRC) PART(MTHLY) SCHPTH(*DFT) SCOPE(*NORMAL) FORCE
                                   (*NO) BLDMODE(*COND) SAVLST(*NO)
SMITH        12/22/95   10:18:11   PRMPART PROJ(PAYROLL) GROUP(SMITH) TYPE(DDSSRC) PART(MTHLY)
SMITH        12/22/95   10:19:45   DLTPART PROJ(PAYROLL) GROUP(SMITH) TYPE(FILE) PART(MTHLY)
TREMBLAY     12/22/95   10:20:44   BLDPART PRJ(PAYROLL) GRP(TEST) TYPE(*ALL) PART(*ALL) SCHPTH(*DFT) SCOPE(*NORMAL) FORCE(*NO
                                   ) BLDMODE(*COND) SAVLST(*NO)
SMITH        12/22/95   10:22:13   ADDADMTYPE TYPE(ZPASSRC) SYSTYPE(*MBR) DFTSRCF(QRPGSRC) DFTRCDLEN(00092) ALWI NC(*NO) INCL
                                   UDES(*NONE)
SMITH        12/22/95   10:23:59   ADDADMLANG TYPE(ZPASSRC) LANG(RPG38) BLDCMD(*NONE) BLDOUTTYPE(*NONE ) BLDOUTLANG(*INPUT) S
                                   PLFNAME(*INPUT)
             * * * * * E N D  O F  L I S T I N G  * * * * *
```

Figure 62. A Sample Report Produced by the PRTPRJLOG Command

If you want to see more than the day's activities, you can indicate a FROMDATE and a TODATE on the PRTPRJLOG command.

The special values *CURRENT and *BEGIN are supported on the FROMDATE parameter. The default is to use *CURRENT, that is, today's date. If you specify *BEGIN, you will see all project log entries from the beginning of the project log to the date specified on the TODATE parameter. You can also specify a particular date on the FROMDATE parameter. The date must be in the format specified by the system values QDATFMT and, if separators are used, QDATSEP.

The special value *CURRENT is supported on the TODATE parameter. It is the default. You can also specify a particular date on the TODATE parameter. This date must be in the format specified by the system values QDATFMT and, if separators are used, QDATSEP.

You can also print entries according to user profiles. The default for the USRPRF parameter is *ALL. This causes a report that contains entries for all users with access to the project to be printed.

If you want to see only your own activity within the project, specify USER(*USRPRF). If you want to see the activity of a specific person, specify the user profile name on the USRPRF parameter.

Use the STRINGS parameter if you want to indicate the criteria on which the audit records are to be selected. You may use this function to limit the audit log report by specifying a string contained in commands. For example, you can specify a specific command name, or a specific group and part name used in commands.

On the STRINGS parameter, you can specify up to four strings which appear in the audit log. Each string can contain up to 32 characters. All comparisons are done in upper case.

Each audit log entry selected will contain **all** the strings specified.

Note: This will have an effect of the AND operation on all strings.

If you want to print a report containing any one of the string, then you need to issue the PRTPRJLOG commands separately by specifying each string.

Examples: This example illustrates how to print a report with the day's activity for the user profile ROBERTS.

Using the PRTPRJLOG command

```
PRTPRJLOG PRJ(PAYROLL) FROMDATE(*CURRENT) TODATE(*CURRENT) USRPRF(ROBERTS)
          STRINGS("RPGSRC")
```

Using the Programming Development Manager utility

Select option 36 (Print log) on the Work with Projects Using PDM display.

If you do not want to print the report produced by the PRTPRJLOG command, you can direct the output to an output file by choosing OUTPUT(*OUTFILE). You must specify a library and file to receive the output. The following command is an example of how to do this.

```
PRTPRJLOG PRJ(PAYROLL) FROMDATE(*CURRENT) TODATE(*CURRENT) USRPRF(ROBERTS)
          OUTPUT(*OUTFILE) OUTFILE(*LIBL/OUTFILE) OUTMBR(*FIRST)
```

The record format of the output file is the same as that used in the system-supplied database file QALYPRJLOG in library QADM.

Recovering When a User Leaves the Project

It is not uncommon for employees to move to other jobs within an organization or to leave the organization entirely. If a developer leaves part way through the application development cycle, the administrator needs to verify that this person's work is complete. Depending on the circumstances in which the developer leaves, the work underway can be assigned to another developer, completed by the administrator, or deleted from the project hierarchy.

Listing the Parts in the Development Group

The first step is to assess how much work is still pending in the developer's development group. Use the Query Part (QRYPART) command to determine what information exists in this group.

Example: The following command creates a report that lists all the parts within the development group ROBERTS.

```
QRYPART PRJ(PAYROLL) GRP(ROBERTS) TYPE(*ALL) PART(*ALL) OUTPUT(*PRINT)
        SCAN(*NO)
```

The SCAN(*NO) option is specified so that only those parts in the ROBERTS group, and not all the parts in the search path for this group, are listed.

Deleting Parts that Cannot be Promoted

Parts of type PGM can be deleted because they can be recreated by the build process with the BLDPART command. Parts of other types, for example, FILE, CLD, CMD, and MODULE can also be deleted because these parts are not normally promoted. (You can promote parts of type, such as, FILE, CLD, CMD, and MODULE, using the PRMPART command with EXTEND(*YES) parameter and specify their corresponding source parts.) Use the DLTPART command to delete these parts if you wish.

Example: The following command deletes all the parts of type PGM in the ROBERTS development group.

```
DLTPART PRJ(PAYROLL) GRP(ROBERTS) TYPE(PGM) PART(*ALL)
```

The next step is to determine which parts listed in the development group are checked out to the user profile ROBERTS. (In this example, the name of the development group and the name of the user profile are the same.)

For each of the remaining parts in this group, use the Print Part Information (PRTPARTINF) command to determine if the access key is set to ROBERTS. As with the QRYPART command, use the SCAN(*NO) parameter on the PRTPARTINF command to restrict the search to this group.

Example: The following command produces a report that lists information about the part BIWKLY.

```
PRTPARTINF PRJ(PAYROLL) GRP(ROBERTS) TYPE(RPGSRC) PART(BIWKLY)
           OUTPUT(*PRINT) SCAN(*NO)
```

If the report shows that the access key is set to ROBERTS, manual intervention is required. The circumstances in which the developer left the organization determine how to handle the next step. The access key must be changed to one of the following values by using the CHGPARTINF command:

- Set the access key to *NONE so the part may be deleted
- Set the access key to another developer's user profile (transferring the work to that developer)
- Set the access key to your own user profile.

Example: If you feel that the changes in the part should not be kept, use the following command to change the access key to *NONE.

```
CHGPARTINF PRJ(PAYROLL) GRP(ROBERTS) TYPE(RPGSRC) PART(BIWKLY)
           LANG(*SAME) ACCKEY(*NONE)
```

This command releases the access key for the part. The copy in this group can now be deleted with the DLTPART command. Note, however, that whatever changes were made to the part are lost. If the part is a new part that has not yet been promoted, when it is deleted from the group ROBERTS it is removed completely from the project hierarchy. If the part has already been promoted, another developer can check the old part out and change it in his or her development group.

If you feel that the changes in the part should be evaluated, change the access key to your own user profile, or use the CHGPRJUSR command to give another developer update access to the group ROBERTS, and change the access key to that person's user profile. This allows whomever is specified on the access key to change the part.

Examples: The following commands give SMITH the required update access and change the access key for the part BIWKLY so that SMITH can work on it.

```
CHGPRJUSR PRJ(PAYROLL) USRPRF(SMITH) USRTYPE(*SAME) ACCESS(*UPDATE)
          DEVELOPGRP(SMITH ROBERTS)

CHGPARTINF PRJ(PAYROLL) GRP(ROBERTS) TYPE(RPGSRC) PART(BIWKLY)
           LANG(*SAME) ACCKEY(SMITH)
```

You must use the CHGPARTINF command for each part in the development group ROBERTS for which the access key is set to ROBERTS.

Once the changes have been tested, the part can be promoted.

Removing a Project User (RMVPRJUSR)

The next step is to remove the developer's access or enrollment information using the Remove Project User (RMVPRJUSR) command.

Example: The following command removes ROBERTS from the project.

```
RMVPRJUSR PRJ(PAYROLL) USRPRF(ROBERTS)
```

The Application Development Manager/400 feature ensures that there is always one project administrator enrolled in a project. If an attempt is made to remove the only project administrator, an error message is issued.

In the unusual situation where the only project administrator leaves the project, someone must sign onto the AS/400 system with that administrator's user profile or the QSECOFR user profile, and enroll a new project administrator for the project using the ADDPRJUSR command. The new administrator can then use the RMVPRJUSR command to remove the previous administrator's enrollment information.

Deleting the Group (DLTGRP)

The final step is to remove the group from the project hierarchy if it is not shared by other developers.

Example: The following command removes the group ROBERTS.

```
DLTGRP PRJ(PAYROLL) GRP(ROBERTS)
```

The Application Development Manager/400 feature ensures that a group being deleted is empty. For example, if you attempt to delete a group with the DLTGRP command, and it still contains parts whose access key is set to a user assigned to that group, the command fails.

Audit Trail Using Journals

Application Development Manager/400 uses journals to keep the audit trail of all commands of this product. You can print this audit trail using the PRTPRJLOG command. This is achieved using journal QLYJRN in QUSRSYS. Another journal QLYPRJLOG in QUSRSYS is used to journal all transactions to the data store files of the product. These transactions are used for reclaim project.

The journals are sensitive to the system date and time. If due to any reason you change the system date to any future date and time, make sure that no one uses any of the commands of this product until the system date and time is returned to normal. If any one does, then you will not able to use the PRTPRJLOG command as desired. The DSPJRN command internally used by the PRTPRJLOG command does not retrieve any record past the current date (default used by PRTPRJLOG), and hence will not return any records beyond the record containing the future time stamp. This may also happen in QLYPRJLOG journal, and will cause similar problem for reclaim project.

To resolve this problem, you need to:

1. Find and delete the journal receivers containing the bad entries with a future time stamp. If the entries are in the current journal receivers, then note down the names of files being journaled and current journal receiver name (you may use the WRKJRNA command on QUSRSYS/QLYJRN).
2. Issue the ENDJRNPF command for all files being journaled on QUSRSYS/QLYJRN.
3. Delete QLYJRN journal in QUSRSYS.
4. Delete the current journal receiver.
5. Create the journal receiver QLYRCVnnnn, where nnnn is a 4 digit number starting from 0001, with THRESHOLD(10000).
6. Create the QLYJRN journal associated with the QLYRCVNNNN journal receiver.
7. Start journaling the physical files whose names you noted earlier using the STRJRNPF command.

Repeat the above steps 1, and steps 3 to 6 for the journal QLYPRJLOG in QUSRSYS library.

Considerations for Administrators Working as Developers

Project administrators who also work as developers should be aware that there are certain part development commands that must be used with caution. Problems can sometimes arise if project administrators forget that they can produce a broader range of results for certain part development commands than developers can.

Administrators can delete a part from a development group, or from any other group in the project hierarchy, by using the DLTPART command. Administrators must use this command with caution when working as developers.

Administrators can also change the access key for a part by using the CHGPARTINF command. Before changing the access key for a part that is checked out to a developer, administrators should ensure that the developer is aware of their intentions.

Chapter 16. Using the Programming Development Manager Utility

As an alternative to CL commands, you can use the Programming Development Manager utility to carry out part development tasks. This chapter discusses how to work with projects, groups, and parts using the Programming Development Manager utility.

Overview of the Programming Development Manager Utility

The Programming Development Manager displays list the projects, groups, or parts that are available to you, along with the operations that you can perform on them. The operations are presented as numbered options at the top of the display, and function keys at the bottom of the display. To perform an operation, type its number in the Opt field next to the appropriate project, group, or part, and then press Enter, or use the appropriate function key.

To see more options on any display where more options are available, press F23=More options. If you want to repeat an option for all of the parts listed, type the option number and then use F13=Repeat. This repeats the option for all the parts below the first one. To see more function keys on any display where more function keys are available, press F24=More keys.

On some displays, you can enter either a specific name or a generic name. The format for specifying the **generic** value is the same for all displays. If you use the generic value, you can type a partial name qualified by an asterisk (*) to display a specific subset. For example, the generic name can be in one of the following formats:

ABC*
Displays a list of all items that begin with the characters ABC.

***ABC**
Displays a list of all items ending with the characters ABC.

B
Displays a list of all items that have the character B anywhere in the name.

A*C
Displays a list of all items that begin with the character A and end with the character C.

"a*"
Displays a list of all items within quotation marks that start with a.

****ALL**
Displays a list of all items ending with ALL. The double asterisk is needed here, since *ALL is a special value to display a list of all items.

Some prompts have a default value of *ALL, which comes up the first time you use the display. The next time you use it, the input you last entered will appear as the default. To restore the previous value used, press F5=Refresh. For more information on how to use the displays, see *ADTS/400: Programming Development Manager*.

Getting Started

To start the Programming Development Manager utility, enter the STRPDM command. The main menu appears, as shown in Figure 63.

```
                  AS/400 Programming Development Manager (PDM)

Select one of the following:

     1. Work with libraries
     2. Work with objects
     3. Work with members
     4. Work with projects
     5. Work with groups
     6. Work with parts

     9. Work with user-defined options




Selection or command
===> _____________________________________________________________________
_____________________________________________________________________________
F3=Exit       F4=Prompt       F9=Retrieve        F10=Command entry
F12=Cancel    F18=Change defaults
```

Figure 63. AS/400 Programming Development Manager (PDM) Main Menu

Options 4, 5, and 6 appear on your display only if you have the Application Development Manager/400 feature installed.

If you already know the names of both the project and the group that contain the parts you want, choose option 6. If you know the project but need to determine the group, choose option 5. If you need to determine both, use option 4. (Options 1, 2, and 3 are not directly related to the Application Development Manager/400 feature, so are not discussed in this book. There are no options 7 and 8.)

Specifying a Project

You can specify the project you want to work with from the Work with Projects Using PDM display. There are two ways to call this display:

1. From the main menu, choose option 4 (Work with projects). The Specify Projects to Work With display appears, as shown in Figure 64.

```
                        Specify Projects to Work With

Type choice, press Enter.

  Project  . . . . . .   *ALL_____________________________  *ALL, name
                                                           *generic*









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

Figure 64. Specify Projects to Work With Display

   a. In the *Project* prompt, if you know what project or projects you want, enter a specific name or a generic name. If you want to see a list of all projects to which you are enrolled, specify *ALL.

   b. Press Enter. The Work with Projects Using PDM display appears, listing the project or projects you specified on the previous screen. See Figure 65 on page 218.

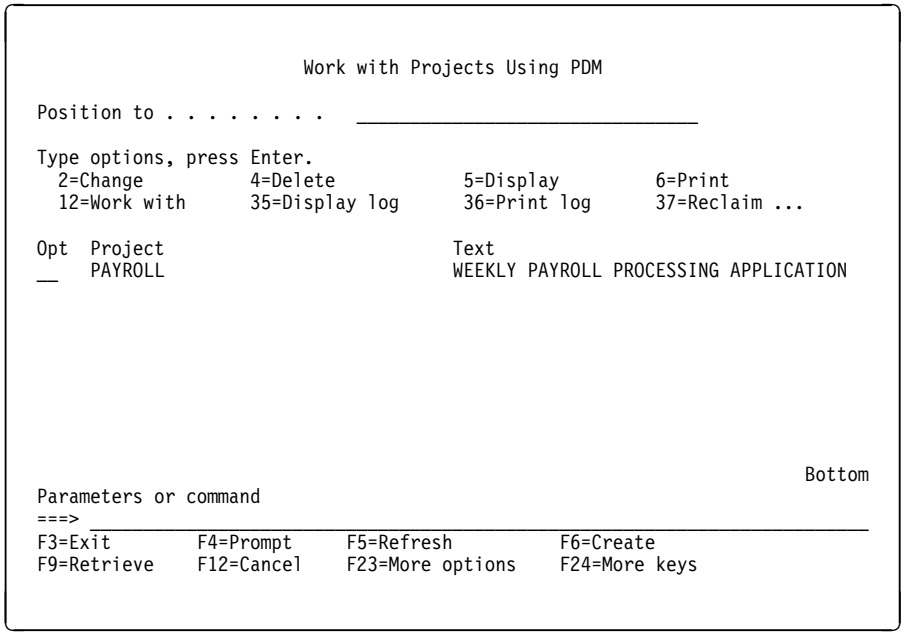

Figure 65. Work with Projects Using PDM Display

Using this display, you can:

- Create, delete, or reclaim projects
- Change, display, or print project information
- Display or print audit log information for a project
- Add, change, remove, display, or print information about the users of a project

For more information on the audit log, refer to Chapter 15, “Securing Application Development Manager/400 Information.”

2. From outside the Programming Development Manager utility, you can use the WRKPRJPDM command to go directly to the Work with Projects Using PDM display, bypassing the main menu and the Specify Projects to Work With display. Type WRKPRJPDM on the command line.

   If you press Enter, the display appears, showing all the projects to which you are enrolled. This is equivalent to specifying *ALL on the Specify Projects to Work With display. If you press F4=Prompt instead, you will first be presented with a prompt. Use the default value *PRV (to indicate you want to use what you specified in your previous session), *ALL, *generic*, or type a name, and press Enter. The Work with Projects Using PDM display then appears, showing only the projects you specified.

On the Work with Projects Using PDM display, type 12 (Work with) next to the project whose groups you want to see. This option presents you with the Work with Groups Using PDM display shown in Figure 67 on page 220.

Specifying a Group

You can specify the group you want to work with from the Work with Groups display. There are three ways to call this display:

1. From the main menu, choose option 5 (Work with groups) to bring up the Specify Groups to Work With display, as shown in Figure 66.

```
                        Specify Groups to Work With

Type choices, press Enter.

  Project  . . . . . .   PAYROLL_____________________________   Name

  Group  . . . . . . .   *ALL_______________________________   *ALL, name









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

Figure 66. Specify Groups to Work With Display

   a. In the *Project* prompt, enter the name of the project that contains the group or groups you want to see. In the *Group* prompt, if you know what groups you want, enter a specific name or a generic name. If you do not know the name of the group, press F4=Prompt to get a list of all the groups in the project. From this list you can select the groups you want to work with.

   b. Press Enter. The Work with Groups Using PDM display appears, listing the groups you specified on the previous screen. Use F11=Display hierarchy to present the groups in a way that shows the level of each in relationship to the rest of the project hierarchy. The target group is at the top of the list. See Figure 67 on page 220.

      If you used F11 in a previous session, the groups are presented in a hierarchical fashion when the Work with Groups Using PDM display appears.

```
                        Work with Groups Using PDM

Project . . . . . .   PAYROLL_____________________

Type options, press Enter.
  2=Change         4=Delete        12=Work with      14=Build
  25=Find string   29=Check in     30=Promote        38=Import ...

Opt  Level  Group
__     01    MASTER
__     02      TEST1
__     03        DEVELOPER1
__     03        DEVELOPER2
__     02      TEST2
__     03        DEVELOPER3


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

Figure 67. Work with Groups Using PDM Display

The following additional options are accessed on the Work with Groups Using PDM display by pressing F23=More options.

```
Type options, press Enter.
  39=Export         45=Add project library list ...
```

Figure 68. Options when F23 is Pressed

Also, the following additional function keys are accessed on the Work with Groups Using PDM display by pressing F24=More keys. Press F24=More keys once to obtain the keys illustrated in Figure 69, and twice to obtain the keys illustrated in Figure 70.

```
Parameters or command
===> ____________________________________________________________________________
F11=Display groups only   F12=Cancel  F13=Repeat   F14=Display text
F18=Change defaults       F23=More options         F24=More keys
```

Figure 69. Function Keys when F24 is Pressed Once

```
Parameters or command
===> ____________________________________________________________________________
F20=Remove project library list          F21=Print list
F23=More options                         F24=More keys
```

Figure 70. Function Keys when F24 is Pressed Twice

From this display, you can do the following tasks:

- Change group information
- Delete a group from a project hierarchy
- Build, check in, promote, import, export, or find a string in all parts in a group

2. From the Work with Projects Using PDM display shown in Figure 65 on page 218, type 12 (Work with) next to the project whose groups you want to see. The Work with Groups Using PDM display appears, listing all the groups within this project.
3. From outside the Programming Development Manager utility, you can use the WRKGRPPDM command to go directly to the Work with Groups Using PDM display. This way, you bypass the main menu and the Specify Groups to Work With display.

   Type WRKGRPPDM on the command line. If you press Enter, the display appears, showing all the groups contained in the last project you accessed. (If this is your first session, no project can be determined, and you receive an error message.)

   If you press F4=Prompt instead, you are first presented with prompts. Use the default value *PRV to indicate you want to use what you specified in your previous session (this assumes this is not your first session), or type a name, and press Enter. The display appears, showing only the groups you specified.

On the Work with Groups Using PDM display, type 12 (Work with) next to the group whose parts you want to see. Developers have read access to all the groups in their project.

Working with Parts

You can specify the part you want to work with from the Work with Parts display. There are three ways to call this display:

1. From the main menu, choose option 6 (Work with parts) to bring up the Specify Parts to Work With display, as shown in Figure 71.

```
                        Specify Parts to Work With

Type choices, press Enter.

  Project  . . . . . .   PAYROLL_____________________________   Name

  Group  . . . . . .    DEVELOPER1____________________________   Name

  Type . . . . . .      *ALL_____                               *ALL, type
                                                                *generic*

  Part . . . . . .      *ALL_____                               *ALL, name
                                                                *generic*

  Language . . . . . .  *ALL_____                               *ALL, *NONE
                                                                language

  Part List  . . . . .  *NONE____                               *NONE, name



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

Figure 71. Specify Parts to Work With Display

In the *Project* and *Group* prompts, enter the names of the project and group that contain the parts you want. In the *Type, Part,* and *Language* prompts, either enter specific or generic names for the type, part name, and language of the parts you want, or enter *ALL if you want to see all parts.

In the *Part List* prompt, either enter a specific name of a part-list part or *NONE.

If you specify *NONE and press Enter, the Work with Parts Using PDM display appears. See Figure 72 on page 223.

If you specify a specific name of a part-list part and press Enter, the Work with Part List Parts Using PDM display appears. See Figure 79 on page 226.

Alternately, from the Work With Parts Using PDM display you can specify the PL user-defined option to go to the Work With Parts in Part List Using PDM display.

```
PL  WRKPARTPDM PRJ(&ZP) GRP(&ZG) TYPE(*ALL) PART(*ALL) LANG(*ALL) PLIST(&N)
```

2. From the Work with Groups Using PDM display described above, type 12 (Work with) next to the group whose parts you want to see. The Work with Parts Using PDM display appears.
3. From outside the Programming Development Manager utility, you can use the WRKPARTPDM command to go directly to the Work with Parts Using PDM display, bypassing the main menu and the Specify Parts to Work With display.

   Type WRKPARTPDM on the command line. If you press Enter, the display immediately appears, showing all the parts contained in the group you last accessed. (If this is your very first session, no project and group can be determined, and you receive an error message.)

   If you press F4=Prompt instead, you are first presented with prompts. In the *Project* and *Group* prompts, type *PRV to indicate you want to use what you specified in your previous session (this assumes this is not your first session), or enter a specific name. In the *Type, Part,* and *Language* prompts, either type specific or generic names for the type, part name, and language of the parts you want, or enter *ALL if you want to see all parts. Press Enter. The display then appears, showing all the parts contained in the group you specified.

Working with Parts Using PDM

The Work with Parts Using PDM display shows parts that exist in the search path specified on the Change Session Defaults display. If the same part exists in more than one group, you see only the part that exists in the lowest group in the search path. You do not see more than one copy of the same part listed on the Work with Parts Using PDM display. Parts are listed in the following order:

1. Source parts are listed in alphabetical order by type. Parts of each type are listed in alphabetical order by their part name.
2. Include parts.
3. Build options parts.
4. Search path parts.
5. All other object types.
6. User-defined types are listed in alphabetical order by type.

```
                        Work with Parts Using PDM

Project . . . . . .   PAYROLL_____________________________
Specified group . . . .   DEVELOPER1_____________________
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
__   BIWKLY      RPGSRC      RPG         MASTER
__   MNTHLY      RPGSRC      RPG         MASTER
__   OVRTM       RPGSRC      RPG         TEST
__   WKLY        RPGSRC      RPG         DEV1


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

Figure 72. Work with Parts Using PDM Display

In Figure 72, the group DEVELOPER1 and search path *DFT were specified; however, all parts are shown as existing in the group MASTER. This means that there are currently no parts in DEVELOPER1, but parts were found in the group MASTER, which is in the default search path for the group DEVELOPER1.

If you want the Work with Parts Using PDM display to show only those parts that meet certain criteria, press F17=Subset. The Subset Part List display appears as shown in Figure 73.

```
                              Subset Part List

Type choices, press Enter.

  Part . . . . . .   *LY_______                    *ALL, name
                                                   *generic*
  Type . . . . . .   *RPGSRC___                    *ALL, type
                                                   *generic*
  Language . . . . . .   *ALL______                *ALL, *NONE
                                                   language
  Group  . . . . . .   MASTER_____________________  *ALL, name

  From date  . . . . .   01/01/00                   Earliest date

  To date  . . . . . .   12/31/99                   Latest date

  Text . . . . . .   *ALL_____________________________________________
  ________________________


  Part list part . . .   *NONE_____                 *NONE, name

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

Figure 73. Subset Part List Display

In this example, you could specify that you only want to see parts of type RPGSRC whose part names end in the characters LY and that are located in the group MASTER.

Press Enter. The Work with Parts Using PDM display reappears, now showing only those parts you specified. The message at the bottom of the display indicates that the list is a subset of a larger one. See Figure 74.

You could also type dates in the *From date* and *To date* prompts to form a time range to restrict your subset to those parts whose last changed date falls in this range. The first time you use this display, the default values are 01/01/00 and 12/31/99. The next time you use it, the last dates you used in this session are displayed.

```
                          Work with Parts Using PDM

 Project . . . . . .   PAYROLL_____________________
 Specified group . . . .   MASTER_____________________
 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
 __   BIWKLY      RPGSRC      RPG         MASTER
 __   MNTHLY      RPGSRC      RPG         MASTER
 __   WKLY        RPGSRC      RPG         MASTER




                                                                        Bottom
 Parameters or command
 ===> ____________________________________________________________________________
 F3=Exit          F4=Prompt            F5=Refresh           F6=Create
 F9=Retrieve      F10=Command entry    F23=More options     F24=More keys
This is a subsetted list.
```

Figure 74. A Subset of a list of Parts

If you want to print the list of parts, use F21=Print list. The report is spooled to the output queue of the print device for this job. The printed report gives the project and group in the report header and lists the following information in the body of the report: part, type, language, group, date, text.

The following additional options and function keys are available on the Work with Parts Using PDM display.

```
 Type options, press Enter.
   17=Change using SDA   18=Change using DFU    19=Change using RLU
   25=Find String        26=Print information   28=Check out  29=Check in ...
```

Figure 75. Options when F23 is Pressed Once

```
Type options, press Enter.
  30=Promote    39=Export   45=Add project library list   50=Convert
  52=Work with archived members         54=Compare        55=Merge ...
```

Figure 76. Options when F23 is Pressed Twice

```
Parameters or command
===> _____________________________________________________________________
F11=Display parts and types       F12=Cancel          F13=Repeat
F14=Display date and text         F23=More options    F24=More keys
```

Figure 77. Function Keys when F24 is Pressed Once

```
Parameters or command
===> _____________________________________________________________________
F15=Sort date     F16=User options    F17=Subset       F18=Change defaults
F20=Remove project library list       F21=Print list   F24=More keys
```

Figure 78. Function keys when F24 is Pressed Twice

Function keys F11, F14, and F15 can be particularly useful when you are using this display. They allow you to display different information about the parts, such as date and text instead of language and group, multiple columns of parts, or you can sort by date or by name.

Working with Parts in Part List Using PDM

The Work with Parts in Part List Using PDM display lets you work with parts in a part-list part in a specified project and group. If the specified part-list part is not found in the group specified, the default project hierarchy is searched.

The Work with Parts in Part List Using PDM display appears when a name of a part-list is specified on the Specify Parts to Work With display. This display is similar to the Work with Parts Using PDM display. Any operation that you can perform on the Work with Parts Using PDM display, can also be performed on this display.

When any Application Development Manager/400 command is used by entering an option in front of a part on this display, and if the command contains the part-list (PARTL) parameter, it is automatically prefilled with the part-list part name you are working with.

```
                    Work with Parts in Part List Using PDM

Project . . . . . .   PAYROLL_____________________________
Specified group . . . .   DEVELOPER1_____________________
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
__   BIWKLY      RPGSRC      RPG         TEST1
__   BIWKL2      RPGSRC      RPG         DEVELOPER1
__   MNTHL2      RPGSRC      RPG         MASTER
__   OVRT2       RPGSRC      RPG         MASTER
__   WKL2        RPGSRC      RPG         MASTER
__   PAY000001   PARTL       *NONE       DEVELOPER1

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

Figure 79. Work with Parts in Part List Using PDM Display

Note that it will take longer to display or refresh the Work with Parts in Part List Using PDM display. Keeping your part-list short will keep this time to a minimum.

Developing Parts

This section presents a sample scenario that leads you through some actions you can perform using the Programming Development Manager utility:

- Checking a part out
- Changing a part
- Copying a part
- Renaming a part
- Converting a part
- Comparing a part
- Merging a part
- Deleting a part
- Building a part
- Testing a part
- Checking a part in
- Finding a string in a part
- Working with archived members
- Promoting a part
- Changing session defaults
- Creating and working with user-defined options

The example assumes that you are working from the Work with Parts Using PDM display shown in Figure 72 on page 223 and that you have update access to the groups DEVELOPER1 and DEVELOPER2. The four parts you will be working with are called RPGSRC BIWKLY, RPGSRC MNTHLY, RPGSRC WKLY, and RPGSRC OVRTM. They exist in the group MASTER in the project hierarchy shown in Figure 80 on page 227.

To work through the actions described here on your own workstation, use appropriate parts and groups from your project.

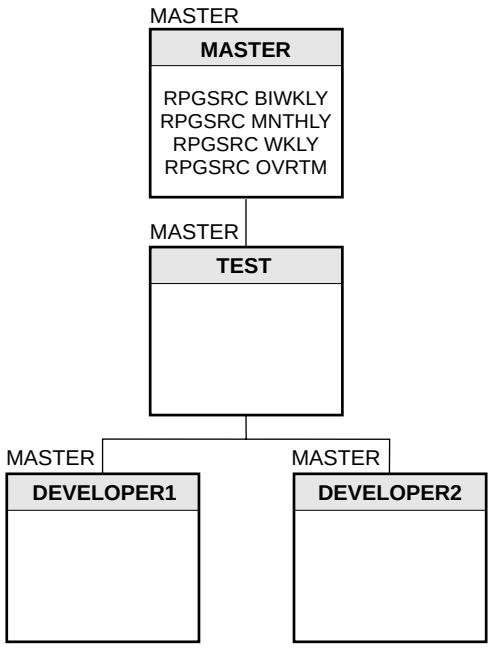

Figure 80. Project Hierarchy Used in the Sample Scenario

Checking a Part Out

Before you check a part out to your development group, you may want to look at it to ensure that it is the right one. To do this, type option 5 (Display) next to its name.

In this example, you are checking out the part RPGSRC BIWKLY to the group DEVELOPER1. This is the group that must be specified in the *Specified group* field. Type 28 (Check out) beside its name. The display now shows this part as being checked out to DEVELOPER1, and a message appears at the bottom of the display to confirm this. (See Figure 81 on page 228.) The part is now locked to your user profile and group.

```
                          Work with Parts Using PDM

  Project . . . . . .   PAYROLL_____________________________
  Specified group . . .   DEVELOPER1_____________________
  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
  __   BIWKLY      RPGSRC      RPG         DEVELOPER1
  __   MNTHLY      RPGSRC      RPG         MASTER
  __   OVRTY       RPGSRC      RPG         MASTER
  __   WKLY        RPGSRC      RPG         MASTER



                                                                      Bottom
  Parameters or command
  ===> ____________________________________________________________________________
  F3=Exit          F4=Prompt             F5=Refresh            F6=Create
  F9=Retrieve      F10=Command entry     F23=More options      F24=More keys
Part BIWKLY checked out to you.
```

Figure 81. Work with Parts Using PDM Display

Changing a Part

Type 2 (Change) next to the part BIWKLY RPGSRC to change it. If the part is not already checked out to you, it will be checked out. Since the part is already in the specified group, no copying is done.

Since the part is a source part, you are placed in an SEU editing session. When you have finished making your changes and have saved them, the Work with Parts Using PDM display reappears, with a message indicating that the part was successfully changed.

If the part you want to change is a non-source part, you are presented with the appropriate AS/400 change command instead of an SEU session.

Instead of option 2, use option 17, 18, or 19 to change a part using SDA, DFU, or RLU:

- If the part is of type DDSSRC with the language attribute of DSPF, use option 17 (Change using SDA)
- If the part is a data file, use option 18 (Change using DFU)
- If the part is of type DDSSRC with the language attribute of PRTF, use option 19 (Change using RLU)

Copying a Part

To copy the part BIWKLY from the group DEVELOPER1 to the group DEVELOPER2, type 3 (Copy) next to it. The Copy Parts display appears as shown in Figure 82 on page 229.

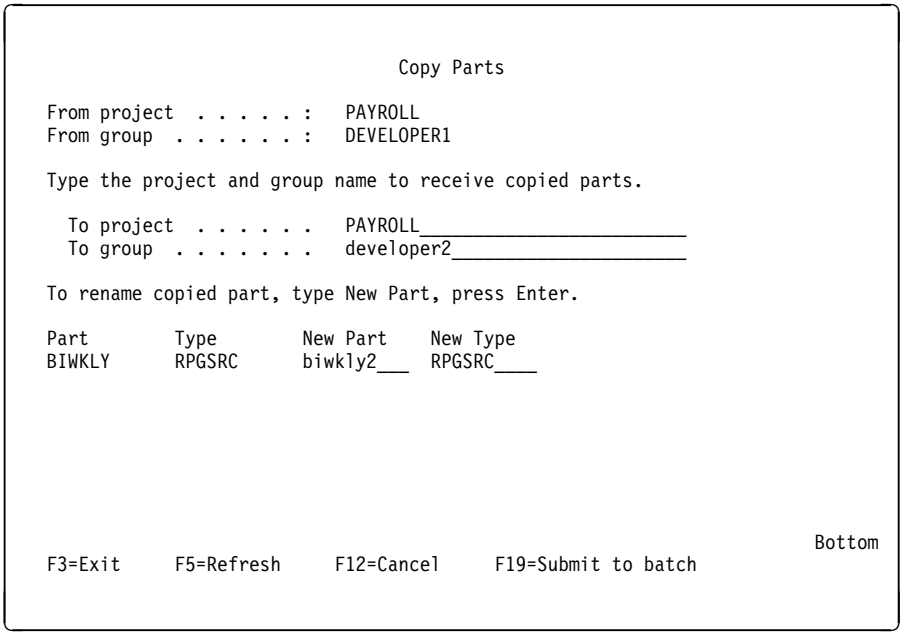

Figure 82. Copy Parts display with a New Group and Part Specified

The defaults for the *To project* and *To group* prompts are the same values that you see in the *From project* and *From group* prompts. If you want to copy the parts to a different group or project, change the information in these fields. You must have update access to the group to which you are copying the parts.

The *Part* and *Type* fields identify the parts you are copying. The defaults for the *New Part* and *New Type* fields are the same values that you see in the *Part* and *Type* fields. If you want to rename the copied part or give it a different type, change the information in these fields.

In Figure 82, the *To group* prompt has been changed from DEVELOPER1 to developer2, and the *New Part* prompt from BIWKLY to BIWKLY2. Press Enter. The part is copied and renamed.

A message appears at the bottom of the display to indicate whether the copy operation was successful. If you tried to copy a part that exists in the promote path of the group you want to copy it to, and if you did not specify a new name for it, the copy fails. Any parts that failed to be copied are highlighted in reverse image.

Renaming a Part

On the Work with Parts Using PDM display, type 7 (Rename) beside the part you want to rename within the same project and group.

The Rename Part display lists the part you want to rename. Type the new name in the *New Part* field, and press Enter. You must have update access to the group containing the part being renamed.

Converting a Part

Type 50 (Convert part) beside the part you want to convert.

The Convert Part display lists the part you want to convert. You cannot change the information in the *Project* and *Group* fields because only part types within the same project and group can be converted. Type the new type name in the *To type* field, and press Enter. The part is converted and its type is renamed.

Comparing a Part

On the Work with Parts Using PDM display, type 54 (Compare part) beside the part you want to compare.

The Compare Part (CMPPART) display allows you to specify the project, group, type and part of the old part that you want to compare. You may also specify various options used in the comparison. The findings of the comparison is placed in a report that you can either display on your workstation or is spooled to your print device.

Merging a Part

On the Work with Parts Using PDM display, type 55 (Merge part) beside the part you want to be the merge target.

The Merge Part (MRGPART) display allows you to specify the maintenance group, maintenance type, maintenance part, root group, root type, and root part that you want to merge.

Deleting a Part

Suppose you copied the wrong part. Type 4 (Delete) next to BIWKLY2 to delete it. The Confirm Delete of Parts display appears, as shown in Figure 83.

```
                          Confirm Delete of Parts

Project . . . . . :   PAYROLL
Group . . . . . :   DEVELOPER2

Press Enter to confirm your choices for Delete.
Press F12=Cancel to return to change your choices.

Part        Type        Language    Group
BIWKLY2     RPGSRC      RPG         DEVELOPER2

                                                                      Bottom
F12=Cancel         F19=Submit to batch
```

Figure 83. Confirm Delete of Parts Display

This display lists the parts that you indicated you want to delete. Press Enter to delete the part, or F12=Cancel.

If you are deleting many parts, you can use F19=Submit to batch to submit all the delete operations to multiple batch jobs. This lets you work on another task while the parts are being deleted.

Building a Part

Type 14 (Build) beside the part, or parts, you want to build.

If you are making many changes to just one part and do not want dependent parts to be compiled, press F18=Change defaults on the Work with Parts Using PDM display to reach the Change Defaults display, and then change the *Build scope* prompt to 2=Limited instead of the default (1=Normal). See “Changing Session Defaults” on page 233 for more information.

After you have built your part, you can check the build report for build warning messages, if any. The build report is described in Figure 49 on page 155. The build messages are listed in Appendix E, “Build Report Messages.”

Testing a Part

After a part has been built successfully, you should test it. You can use the ADDPRJLIBL command to add the libraries for the project to the beginning of the user portion of the session library list. See “Adding Project Libraries to the Library List” on page 180 for a description of the ADDPRJLIBL command.

Option 16 (Run) will run the part. This option is valid for parts of type PGM, REXXSRC, and CMD.

You can use two user-defined options that add or remove project libraries from the AS/400 library list. For a description of them, see “Adding and Removing Project Libraries” on page 238. Option 45 (Add project library list) and Option 20 (Remove project library list) from both the Work with Groups and Work with Parts displays add or remove project libraries from the library list.

Checking a Part In

After you have successfully created or changed a part, built it, and run it, you would check it back in so that it is available to other developers who have update access to the same group. To check it in, type 29 (Check in) next to its name. You receive a message indicating that the part has been successfully checked in.

Promoting a Part

When you finish working with the part, type 30 (Promote) to promote it to the next group higher in the project hierarchy.

You receive a message indicating that the part has been successfully promoted, and the *Group* field is updated to show the name of the group in which the part now resides. If you had not already checked the part in, this is done automatically before it is promoted.

Finding a String in a Part

The Find String Part (FNDSTRPART) command allows you to search for character or hexadecimal strings in Application Development Manager/400 parts containing source or file data. If a match is found for the string, you can use any Programming Development Manager option or one of your own user-defined options on the part. The FNDSTRPART command is similar to the Find String PDM (FNDSTRPDM) command; however, it can only be used from the Work with Groups Using PDM and Work with Parts Using PDM displays.

You can use option 25=Find string from these two displays, or you can use the FNDSTRPART command from outside the Programming Development Manager utility. From the Work with Groups display, all the parts in the specified group are searched. From the Work with Parts display, only the specified part is searched. The value you specify on the *Scan hierarchy* prompt on the Change Defaults display determines whether the project hierarchy will be searched. See "Changing Session Defaults" on page 233 for the *Scan hierarchy* prompt values. (For information about the FNDSTRPDM command, see *ADTS/400: Programming Development Manager*.)

Example: The following command searches for the character string 'ADDR' from column 2 to 4 in all parts of all types in the project PAYROLL, starting in the group DEVELOPER1. When the string is found, the CHECKOUT command is prompted so that you can edit the part that contains the 'ADDR'. A list of the parts that contain the character string is printed. The first two records containing the string are printed in character format. The string is not marked. The record is truncated if it is longer than the length of the line.

```
FNDSTRPART STRING('ADDR') PRJ(PAYROLL) GRP(DEVELOPER1) TYPE(*ALL)
           PART(*ALL) OPTION(*CHG *PROMPT) COL(2 4) PRTPARTLST(*YES)
           PRTRCDS(2 *CHAR *NOMARK *TRUNCATE) SCAN(*YES)
```

For a detailed explanation of each prompt on the Find String display or each field on the FNDSTRPART command:

- Press F1=Help on the Find String display on any area of the display for which you want more information
- Prompt the FNDSTRPART command (F4=Prompt) and press F1=Help on any area of the display for which you want more information

Working with Archived Members

On the Work with Parts Using PDM display, type 52 (Work with archived members) beside the source part that you want to work with.

The Work with Members Using PDM display (Figure 84 on page 233) lists the archived members for that part. These are the members which are created when you specify ARCHIVE(*YES) while changing or importing this part to this group or promoting it to this group.

To restore or roll back an archived member, type IM (Import Part) beside the part that you want rolled back.

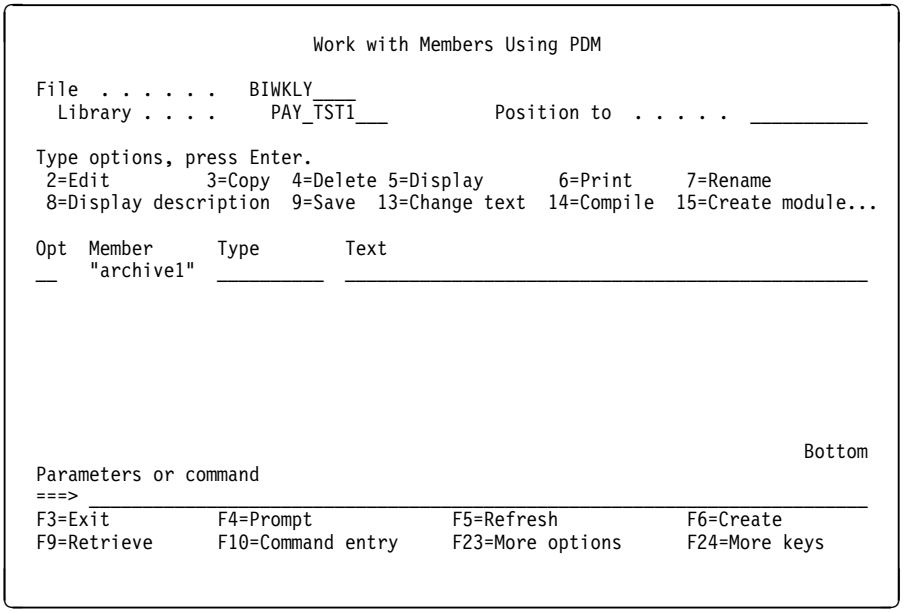

Figure 84. Work with Members Using PDM Display

Note: If you change any session default while working with archived members, those new changes will not be remembered. When you return to the Work With Parts Using PDM display, your session defaults will not be any different than they were before you typed option 52 (Work with archived members).

Changing Session Defaults

This section describes the *Scan hierarchy, Search path, Build scope, Refresh part list,* and *Display source parts only* prompts on the Change Defaults display. They allow you to indicate whether the project hierarchy should be searched for the part, what search path to use to find the part, and the build scope to be used to build the part. To change them, you must change your session defaults.

Use F18=Change defaults from the Work with Parts Using PDM display to get to the Change Defaults display, as shown in Figure 85 on page 234 and Figure 86 on page 234.

```
                              Change Defaults

Type choices, press Enter.

  Object library . . . . . .   *SRCLIB___   Name, *CURLIB, *SRCLIB
  Replace object . . . . . .   N            Y=Yes, N=No
  Compile in batch . . . . .   Y            Y=Yes, N=No
  Run in batch . . . . . . .   N            Y=Yes, N=No
  Save session defaults  . .   Y            Y=Yes, N=No
  Save/Restore option  . . .   1            1=Single, 2=All
  Job description  . . . . .   QBATCH____   Name, *USRPRF, F4 for list
    Library  . . . . . . . .     *LIBL_____ Name, *CURLIB, *LIBL
  Change type and text . . .   Y            Y=Yes, N=No
  Option file  . . . . . . .   TESTFILE__   Name
    Library  . . . . . . . .     *LIBL_____ Name, *CURLIB, *LIBL
  Member . . . . . . . . . .   TESTMBR___   Name
  Full screen mode . . . . .   N            Y=Yes, N=No
  Scan hierarchy . . . . . .   Y            Y=Yes, N=No
  Search path  . . . . . . .   *DFT______   Name, *DFT
  Build scope  . . . . . . .   2            1=Normal, 2=Limited, 3=Extended,
                                            4=Direct Chain
                                                                       More...
F3=Exit     F4=Prompt     F5=Refresh     F12=Cancel
```

Figure 85. Change Defaults Display—Part 1

```
                              Change Defaults

Type choices, press Enter.

  Log option commands  . . .   N            Y=Yes, N=No
  Exit lists on ENTER  . . .   Y            Y=Yes, N=No
  Refresh part list  . . . .   Y            Y=Yes, N=No
  Display source parts only  .   Y          Y=Yes, N=No
```

Figure 86. Change Defaults Display—Part 2

The following list describes the *Scan hierarchy, Search path,* and *Build scope* prompts on the Change Defaults display. The *Scan hierarchy* and *Search path* prompt values are used by any option that calls a command with the SCAN and SCHPTH parameters. See “How Search-Path Parts Are Processed” on page 116 for an explanation of how the scan and search path parameters work together to search for a part.

The value specified on the *Search path* prompt is validated at the time you choose an option on a display that uses a search path. You may need to change the value for the particular task you are performing.

Scan hierarchy

This value determines which parts appear on the Work with Parts Using PDM display and which groups appear on the Work with Groups Using PDM display. It is also used by the various options chosen on those displays. The default is Y=Yes, which means the groups in the default search path are searched for parts.

N=No specifies that only the group specified is searched.

Search path

This value determines which search path an Application Development Manager/400 option uses and shows the parts contained in the groups for a specified search path. The default is *DFT, or you can specify the name of a search path part.

If you specify a particular search path, you can see versions of parts that are not available through the default search path. If you wish to find parts using a search path specified in SCHPTH QDFT part, you must specify QDFT on this prompt. If you specify *DFT on this prompt, and a SCHPTH QDFT exists in the hierarchy, it is not used to determine the list of parts for the Work with Parts Using PDM and Work with Parts in Part List Using PDM displays. However, all Application Development Manager/400 part development commands use SCHPTH QDFT.

If you specify a search path part that searches across projects, the Programming Development Manager utility ignores any groups that do no belong to the project specified in the search path. The parts listed on the Work with Parts Using PDM and the Work with Parts in Part List Using PDM displays are automatically refreshed when the search-path value is changed on this display.

The specified search path does not determine the list of groups displayed on the Work with Groups Using PDM display.

Build scope

The default is 1=Normal. This prompt corresponds to the SCOPE parameter on the BLDPART command. The part specified is built, as are all of the other parts on which it depends, and on which they depend, and so on.

See “Setting the Build Scope” on page 148 for more information.

Refresh part list

This prompt specifies whether or not a part list is refreshed after processing options on the Work with Parts display. For example, if you specify Y=Yes on this prompt, and you delete a part from the displayed part list, the part list is displayed again without the deleted part.

Y=Yes specifies that a part list is refreshed after processing an option on the Work with Parts display.

N=No specifies that a part list is not refreshed after processing an option on the Work with Parts display.

Display source part only

This prompt specifies that all the parts in a specified project and group are displayed in a part list or only source parts in a specified project and group are displayed in a part list.

Y=Yes specifies that only source parts in the specified project and group are displayed in a part list.

N=No specifies that all parts in the specified project and group are displayed in the part list.

The *Object library, Replace object,* and *Change type and text* prompts are ignored except when you use user-defined options. However, all defaults can be changed at any time, regardless of whether you are using the Application Development Manager/400 feature.

User-Defined Options

In addition to the options available on all displays, you can also create your own, called **user-defined options.** These are useful because they allow you to carry out operations you do frequently simply by typing an option on a display.

The command you choose to correspond to your user-defined option can be any AS/400 system or user command. It can contain parameter variables so that the command can be performed on an item in a list. This saves you from specifying an option (that corresponds to a command), prompting the option, and specifying values for the parameters; and it also allows you to perform the command you choose on an item in a list.

Creating a User-Defined Option

The following example creates a user-defined option named EP that calls a CL program named EDITPART. This is the CL program found in "Retrieving Part Information (RTVPARTINF)" on page 100. The program checks a part out, retrieves the AS/400 information necessary to edit the part, starts an editor, and then checks the part back in.

Once this user-defined option is created, you use it from the Work with Parts display:

1. If you are on the Work with Parts Using PDM display, press F16=Work with user options to get to the Work with User-Defined Options display. If you are on the main menu, select option 9 (Work with user-defined options) to bring up the Specify Option File to Work With display. Specify an option file to work with, and press Enter.
2. Press F6=Create on the Work with User-defined Options display. The Create User-Defined Option display appears, as shown in Figure 87.

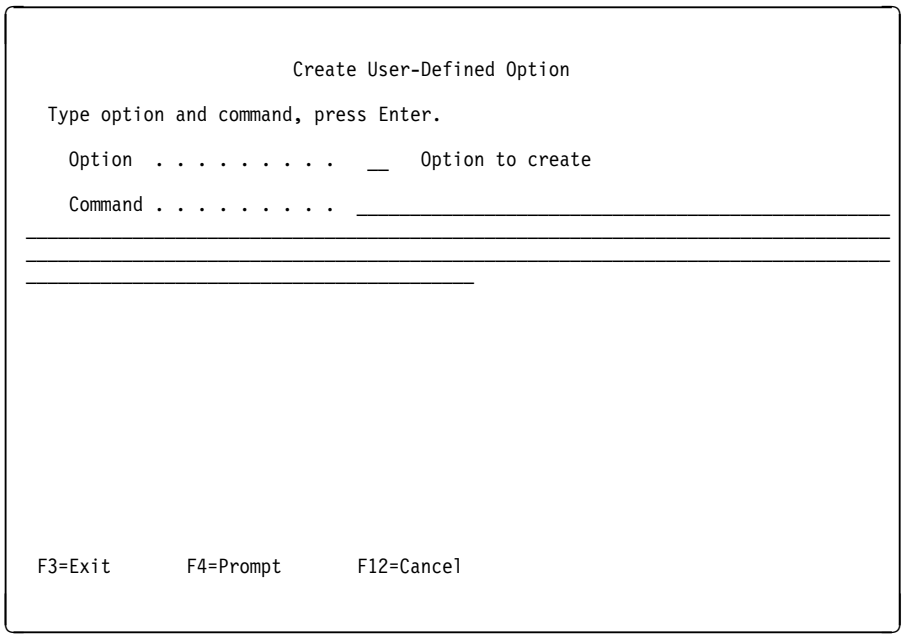

Figure 87. Create User-Defined Option Display

3. In the *Option* prompt, type either 1 or 2 characters to represent the command you want your new option to call. The first character must be a letter; the second one can be either a letter or a number.

   For this example, type EP in the *Option* prompt.

4. In the *Command* prompt, type the command you want to be called when the EP option is selected. If you cannot remember the correct name of the command, press F4=Prompt to display a menu that lets you display all system commands or specific types of commands. If you know the name of the command but want to check its parameters, type the command name, and press F4=Prompt to see its prompt display.

   For this example, type the following command in the *Command* prompt.

   ```
   CALL EDITPART (&ZP &ZG &ZT &ZN)
   ```

   This command will call the EDITPART CL program found in "Retrieving Part Information (RTVPARTINF)" on page 100.

The substitution variables resolve as follows:

&ZP	The name of the project
&ZG	The name of the group
&ZT	The type of the part
&ZN	The name of the part

You can also use the single-character substitution variables &A, &B, &D, &F, &L, &N, &O, &R, &S, &T, and &X to create your user-defined option. They resolve to the AS/400 objects or members corresponding to Application Development Manager/400 projects, groups, or parts listed on the Work-with displays. This allows you to use your current user-defined options to perform Application Development Manager/400 functions. See Appendix D, "Substitution Variables" for a description of all substitution variables.

Working with User-Defined Options

Importing AS/400 Objects or Source Members

There are two system-supplied user-defined options, IO (import object), and IM (import member) to assist you with importing AS/400 objects into the Application Development Manager/400 project hierarchy where they become parts. Using these options with F13=Repeat, you can import many objects at once. The IO, and IM options are included in the system-supplied user-defined option file QAUOUSR in the library QPDA.

Use IO from the Work with Objects display.

```
IO  IMPPART OBJ(&L/&N) OBJTYPE(&T) TYPE(&S) PART(&N) TEXT(&X)
```

Use IM from the Work with Members display.

```
IM  IMPPART OBJ(&L/&F) OBJTYPE(*FILE) MBR(&N) PART(&N) LANG(&S) TEXT(&X)
```

When you choose one of these options, a prompt appears where you can type the name of the project, group, type, and the other information the IMPPART command needs.

Option 38 (Import) from the Work with Groups display provides the same function as the IM and IO user-defined options.

Adding and Removing Project Libraries

Two other system-supplied user-defined options are AP and RP. They are used to add and remove project libraries from the AS/400 library list when you are testing a part. The AP and RP options look like this:

```
AP  ADDPRJLIBL PRJ(&ZP) GRP(&ZG) SCAN(&ZH) SCHPTH(&ZS)

RP  RMVPRJLIBL
```

To use them:

1. Type AP beside any part in the list.

   Project libraries in the search path are added to the AS/400 library list using the ADDPRJLIBL command.

2. Use 16 (Run) option to run a part. When the part has finished, you are returned to the Work with Parts Using PDM display.

3. Type RP beside any part in the list.

   The project libraries are removed from the AS/400 library list using the RMVPRJLIBL command.

   Option 45 (Add project library list) and F20=Remove project library list on the Work with Parts display provide the same function as AP and RP, respectively.

Chapter 17. Working with the VisualAge for RPG Product

The Application Development Manager/400 feature allows you to store and manage VisualAge for RPG applications in Application Development Manager/400 projects on the AS/400 system. To do this, two part types are available: VRPGTXT and VRPGBIN.

VRPGTXT part types are stored in source physical files. The language of the part type will be VRPG. The Application Development Manager/400 feature will treat this part as a single entity. VisualAge for RPG text objects are stored as members with the member types set as their workstation file extension.

VRPGBIN part types are stored in data files. The language of the part type will be VRPG. This part type will include VisualAge for RPG binary objects.

The following Application Development Manager/400 part development commands support these part types:

Change Part (CHGPART)
Change Part Information (CHGPARTINF)
Check In Part (CHKINPART)
Check Out Part (CHKOUTPART)
Compare Part (CMPPART)
Copy Part (CPYPART)
Delete Part (DLTPART)
Display Part (DSPPART)
Export Part (EXPPART)
Import Part (IMPPART)
Print Part Information (PRTPARTINF)
Promote Part (PRMPART)
Query Part (QRYPART)
Reclaim Project (RCLPRJ)
Retrieve Part Information (RTVPARTINF)

The following Application Development Manager/400 part development command supports only the VRPGTXT part type:

Merge Part (MRGPART)

Even though the directory (or component names of VisualAge for RPG objects) on the high performance file system (HPFS) could be very long, the part names of only 10 characters are allowed in the Application Development Manager/400 environment. VisualAge for RPG users can use the part text field to store the additional information.

See the “Bibliography” for a list of VisualAge for RPG publications.

Importing VisualAge for RPG Parts

The only way the VisualAge for RPG parts can be created in Application Development Manager/400 is using the IMPPART command. To import VisualAge for RPG parts, you must specify VRPGBIN or VRPGTXT on the TYPE parameter of the IMPPART command. This is required, since the IMPPART command normally imports individual members as parts from a source file. The VRPGTXT part is imported as a single entity, even though it is stored in a source file.

To import a VisualAge for RPG part, you need to save a component of a VisualAge for RPG project to a library on your AS/400 system using the VisualAge for RPG Client feature. You can use the QTEMP library, if you want to keep it temporarily. Then you can issue the IMPPART command with REPLACE(*YES) to import all VisualAge for RPG source and binary objects separately. The import function will checkout the parts, if required, and check them back in. If a part does not exist, the import function will automatically create it for you.

You cannot change the names of the VisualAge for RPG parts while importing them due to the restrictions imposed by the VisualAge for RPG feature.

Exporting VisualAge for RPG Parts

If you want to restore your VRPG project on your workstation to view or change it, then you must first export the appropriate VRPGTXT and VRPGBIN parts to a library. You can use the QTEMP library as the target library on the EXPPART command. You can then restore the VRPG parts on your workstation using the VisualAge for RPG feature.

If you want to simply view a VRPG project, then you do not need to check out the parts before you issue the EXPPART command. However, if you plan to change the project, you must check the parts out before issuing the EXPPART command.

Building VisualAge for RPG Parts

To build VisualAge for RPG parts, do the following steps:

1. Check out the VisualAge for RPG parts
2. Export the parts
3. Restore the parts to your workstation
4. Invoke the VisualAge for RPG Graphical User Interface (GUI) to build the parts
5. Save the parts to the AS/400 system
6. Import the changed parts to an Application Development Manager/400 project specifying REPLACE(*YES)
7. Check the parts back in

If any of the above mentioned steps fails, then building the VisualAge for RPG parts have also failed.

VisualAge for RPG Part Development

To maintain the integrity of the VisualAge for RPG parts, both the VRPGTXT part and its associated VRPGBIN part should be updated simultaneously. This means that you should always check out, build, export, or import the VRPGTXT part and its associated VRPGBIN part at the same time.

If you delete a VisualAge for RPG part, remember to also delete its associated part. For example, if you delete the VRPGBIN part, you must also delete the VRPGTXT part, and vise versa.

Chapter 18. Working With System/36 and System/38 Applications

This chapter describes:

- Working with the System/36 applications
- Building System/36 applications
- Considerations for the System/38 applications

Working with the System/36 Applications

The Application Development Manager/400 feature directly supports System/36 applications. You may use either the System/36 part types that are shown in Table 20, or you can define your own part types.

Table 20. System/36 Part Types and Language

System/36 part types	Language
CBL36INC	CBL36
CBL36SRC	CBL36
DSPF36SRC	DSPF36
MNU36SRC	MNU36
MSGF36SRC	MSGF36
OCL36SRC	OCL36
RPG36INC	RPG36
RPG36SRC	RPG36
RPT36SRC	RPT36
SRT36SRC	SRT36
TXT36SRC	TXT36

See Appendix B, “Part Types and Their Relationship to Commands” on page 247 for a description of the commands which allow System/36 part types to be specified on the TYPE parameter.

Part types for any System/36 languages not mentioned in the above table, such as BAS36, will have to be defined by you as a user-defined part type.

A part type of *PGM is available with the part languages of CBL36 and RPG36.

The following Application Development Manager/400 commands support the System/36 languages on the LANG parameter:

CHGPARTINF
CRTPART
EXPPART
IMPPART

Building System/36 Applications

You cannot use the BLDPART command to build your System/36 objects, because the System/36 compilers do not return information that the build process needs.

To build your System/36 applications, take the following steps:

1. Set up the library list using the ADDPRJLIBL command.
2. Use the appropriate compiler commands to compile the source parts. To do this, you will need to specify the fully qualified native names of the source part you wish to build. You can retrieve this information using the RTVPARTINF command in a CL program, or using user defined options on the WRKPARTPDM display.
3. The output objects must be created in libraries outside of Application Development Manager/400 control.
4. Import the output objects into the project using the IMPPART command with REPLACE(*YES).
5. Use the RMVPRJLIBL to restore the library list.

Considerations for the System/38 Applications

Although the Application Development Manager/400 feature does not directly support System/38 applications, you can manage these applications by using user-defined part types. Once these types have been defined, you can take advantage of the change management and the version control functions of the Application Development Manager/400 feature.

All other Application Development Manager/400 part commands except the BLDPART command can be used for System/38 source stored in parts with the user-defined part types.

Because the System/38 compilers do not provide build relationship information to the Application Development Manager/400 feature, you will not be able to use the BLDPART command to build these applications directly.

Appendix A. Application Development Manager/400 Control Language Commands

All Application Development Manager/400 control language (CL) commands can be used from an AS/400 command line. AS well, many of these commands can also used by either selecting an option or pressing a function key on one of the following PDM displays:

Work with Projects Using PDM
Work with Groups Using PDM
Work with Parts Using PDM

Table 21 lists all the Application Development Manager/400 CL commands. For information about using PDM displays, see Chapter 16, "Using the Programming Development Manager Utility."

Table 21 (Page 1 of 2). Application Development Manager/400 CL Commands

CL Command	Description
ADDADMLANG[1]	Add a language for a user-defined type
ADDADMTYPE[1]	Add a user-defined type
ADDPRJLIBL	Add project libraries to the AS/400 library list
ADDPRJUSR[2]	Authorize a developer or administrator to the specified project
BLDPART	Build a part
CHGADMACN[1]	Change the action definition for a user-defined type
CHGADMLANG[1]	Change a language for a user-defined type
CHGGRP[2]	Change the characteristics of a group
CHGPART[3]	Change an existing part
CHGPARTINF	Change information about an existing part
CHGPRJ[2]	Change the characteristics of the project
CHGPRJUSR[2]	Change the characteristics of a developer or an administrator authorized to the project
CHKINPART	Release a part that was checked out and make it available for other developers to use
CHKOUTPART	Copy a part to your development group if it does not already exist there, and then lock it to prevent other developers from accessing it
CMPPART	Compare two or more parts
CPYPART	Create a new part based on an existing part
CRTGRP[2]	Create a new group
CRTPART	Create a new part
CRTPRJ	Create a project
CVTPART	Convert a part type and language to another part type and language
DLTGRP[2]	Delete a group and its contents
DLTPART	Delete a part from a specific group
DLTPRJ[2]	Delete a project and its contents

Table 21 (Page 2 of 2). Application Development Manager/400 CL Commands

CL Command	Description
DSPPART	Display a part's contents or characteristics
EXPPART	Copy a part from the project hierarchy into an AS/400 library
FNDSTRPART	Find a particular string in a part or set of parts
IMPPART	Copy an AS/400 member or object and store it in the project hierarchy as an Application Development Manager/400 part
MRGPART	Merge two sets of updates made to one or more parts.
PRMPART	Promote a part to the next group in the hierarchy
PRTADMLANG	Print the language information for a user-defined type
PRTADMTYPE	Print information about a user-defined type
PRTPARTINF	Print information about a particular part
PRTPRJ	Print information about a project
PRTPRJLOG	Print the contents of the project audit log
PRTPRJUSR	Print a list of authorized developers and administrators for the project
QRYPART	Query a part
QRYPRJ	Query a project
RCLPRJ[2]	Reclaim a project after a restore operation or after a system failure
RCVPART	Receive incoming parts from remote AS/400 systems
RMVADMLANG[1]	Remove the language for a user-defined type
RMVADMTYPE[1]	Remove a user-defined type
RMVPRJLIBL	Remove project libraries from the AS/400 library list
RMVPRJUSR[2]	Remove a developer or administrator from a project
RNMPART	Rename a part
RTVPARTINF[4]	Retrieve an AS/400 member or object name for a specific Application Development Manager/400 part
WRKGRPPDM[3]	Work with a list of groups using PDM
WRKPARTPDM[3]	Work with a list of parts using PDM
WRKPRJPDM[3]	Work with a list of projects using PDM

Notes:

1. The ADDADMLANG, ADDADMTYPE, CHGADMACN, CHGADMLANG, RMVADMLANG, and RMVADMTYPE commands require *ALLOBJ authorization.
2. The ADDPRJUSR, CHGGRP, CHGPRJ, CHGPRJUSR, CRTGRP, DLTGRP, DLTPRJ, RCLPRJ, and RMVPRJUSR commands are for project administrator use only.
3. This command is processed in the interactive environment only, as part of a CL program, as part of a REXX procedure, when typed on a command entry prompt, or when called from QCMDEXC.
4. This command is run only as part of a CL program or REXX procedure in either the batch or interactive environments.

Appendix B. Part Types and Their Relationship to Commands

Use the following tables to determine which Application Development Manager/400–supported types can be specified for Application Development Manager/400 commands that contain the TYPE parameter (y=yes, can be specified, <blank>=no, cannot be specified).

Table 22. Application Development Manager/400 Types and How They Relate to CL Commands (Part 1 of 8)

Command	RPGSRC	RPGLESRC	RPGINC	RPGLEINC	CBLSRC	CBLLESRC	CBLINC
BLDPART	y	y	y[1]	y	y	y	y[1]
CHGPART	y	y	y	y	y	y	y
CHGPARTINF	y	y	y	y	y	y	y
CHKINPART	y	y	y	y	y	y	y
CHKOUTPART	y	y	y	y	y	y	y
CMPPART	y	y	y	y	y	y	y
CPYPART	y	y	y	y	y	y	y
CRTPART	y	y	y	y	y	y	y
CVTPART	y	y			y	y	
DLTPART	y	y	y	y	y	y	y
DSPPART	y	y	y	y	y	y	y
EXPPART	y	y	y	y	y	y	y
FNDSTRPART[2]	y	y	y	y	y	y	y
IMPPART	y	y	y	y	y	y	y
MRGPART	y	y	y	y	y	y	y
PRMPART[3]	y	y	y	y	y	y	y
PRTPARTINF	y	y	y	y	y	y	y
QRYPART	y	y	y	y	y	y	y
RNMPART	y	y	y	y	y	y	y
RTVPARTINF	y	y	y	y	y	y	y
WRKPARTPDM	y	y	y	y	y	y	y

Note:

The following items apply to the part types listed in the table:

1. Message files and include parts are not actually built, but can be referred to by parts you are building. The same is true for parts of type BNDDIR; however, they can be built if you specify SCOPE(*EXTENDED) on the BLDPART command. See “Setting the Build Scope” on page 148 for more information.
2. The FNDSTRPART command is valid for a system-supplied or user-defined type stored in a source file and for FILE(PF) parts.
3. In general, objects can be promoted if you specify EXTEND(*YES) on the PRMPART command.

Table 23. Application Development Manager/400 Types and How They Relate to CL Commands (Part 2 of 8)

Command	CBLLEINC	CSRC (C)	CSRC (CLE)	CINC (C)	CINC (CLE)	CMDSRC	CLSRC
BLDPART	y		y		y	y	
CHGPART	y	y	y	y	y	y	y
CHGPARTINF	y	y	y	y	y	y	y
CHKINPART	y	y	y	y	y	y	y
CHKOUTPART	y	y	y	y	y	y	y
CMPPART	y	y	y	y	y	y	y
CPYPART	y	y	y	y	y	y	y
CRTPART	y	y	y	y	y	y	y
CVTPART		y	y		y		
DLTPART	y	y	y	y	y	y	y
DSPPART	y	y	y	y	y	y	y
EXPPART	y	y	y	y	y	y	y
FNDSTRPART[1]	y	y	y	y	y	y	y
IMPPART	y	y	y	y	y	y	y
MRGPART	y	y	y	y	y	y	y
PRMPART[2]	y	y	y	y	y	y	y
PRTPARTINF	y	y	y	y	y	y	y
QRYPART	y	y	y	y	y	y	y
RNMPART	y	y	y	y	y	y	y
RTVPARTINF	y	y	y	y	y	y	y
WRKPARTPDM	y	y	y	y	y	y	y

Note:

The following items apply to the part types listed in the table:

1. The FNDSTRPART command is valid for a system-supplied or user-defined type stored in a source file and for FILE(PF) parts.
2. In general, objects can be promoted if you specify EXTEND(*YES) on the PRMPART command.

Table 24. Application Development Manager/400 Types and How They Relate to CL Commands (Part 3 of 8)

Command	CLLESRC	CLPSRC	DDSSRC	REXXSRC	CLDSRC	TXTSRC	BLDOPT
BLDPART	y	y	y		y		
CHGPART	y	y	y	y	y	y	y
CHGPARTINF	y	y	y	y	y	y	y
CHKINPART	y	y	y	y	y	y	y
CHKOUTPART	y	y	y	y	y	y	y
CMPPART	y	y	y	y	y	y	y
CPYPART	y	y	y	y	y	y	y
CRTPART	y	y	y	y	y	y	y
CVTPART	y	y				y	
DLTPART	y	y	y	y	y	y	y
DSPPART	y	y	y	y	y	y	y
EXPPART	y	y	y	y	y	y	y
FNDSTRPART[1]	y	y	y	y	y	y	y
IMPPART	y	y	y	y	y	y	y
MRGPART	y	y	y	y	y	y	y
PRMPART[2]	y	y	y	y	y	y	y
PRTPARTINF	y	y	y	y	y	y	y
QRYPART	y	y	y	y	y	y	y
RNMPART	y	y	y	y	y	y	y
RTVPARTINF	y	y	y	y	y	y	y
WRKPARTPDM	y	y	y	y	y	y	y

Note:

The following items apply to the part types listed in the table:

1. The FNDSTRPART command is valid for a system-supplied or user-defined type stored in a source file and for FILE(PF) parts.
2. In general, objects can be promoted if you specify EXTEND(*YES) on the PRMPART command.

Table 25. Application Development Manager/400 Types and How They Relate to CL Commands (Part 4 of 8)

Command	SCHPTH	PGM	FILE	CLD	CMD	MSGF	DTAARA	BNDSRC
BLDPART		y[2]	y	y	y	y[1]		y
CHGPART	y	y	y		y	y	y	y
CHGPARTINF	y	y	y	y	y	y	y	y
CHKINPART	y	y	y	y	y	y	y	y
CHKOUTPART	y	y	y	y	y	y	y	y
CMPPART	y		y[5]					y
CPYPART	y	y	y	y	y	y	y	y
CRTPART	y					y	y	y
CVTPART		y						
DLTPART	y	y	y	y	y	y	y	y
DSPPART	y	y	y		y	y	y	y
EXPPART	y	y	y	y	y	y	y	y
FNDSTRPART[3]	y		y					y
IMPPART	y	y	y	y	y	y	y	y
MRGPART	y							
PRMPART[4]	y					y	y	y
PRTPARTINF	y	y	y	y	y	y	y	y
QRYPART	y	y	y	y	y	y	y	y
RNMPART	y	y	y	y	y	y	y	y
RTVPARTINF	y	y	y	y	y	y	y	y
WRKPARTPDM	y	y	y	y	y	y	y	y

Note:

The following items apply to the part types listed in the table:

1. Message file and include parts are not actually built, but can be referred to by parts you are building. The same is true for parts of type BNDDIR; however, they can be built if you specify SCOPE(*EXTENDED) on the BLDPART command. See “Setting the Build Scope” on page 148 for more information.
2. The parts of type PGM with language C cannot be built.
3. The FNDSTRPART command is valid for a system-supplied or user-defined type stored in a source file and for FILE(PF) parts.
4. In general, objects can be promoted if you specify EXTEND(*YES) on the PRMPART command.
5. The CMPPART command is valid for a system-supplied or user-defined type stored in a source file and for FILE(PF) parts, and PARTL parts.

Table 26. Application Development Manager/400 Types and How They Relate to CL Commands (Part 5 of 8)

Command	MODULE	SRVPGM	BNDDIR[1]	PNLSRC	PNLINC	PNLGRP	MENU (*UIM)
BLDPART	y	y	y	y	y	y	y
CHGPART	y	y	y	y	y		y
CHGPARTINF	y	y	y	y	y	y	y
CHKINPART	y	y	y	y	y	y	y
CHKOUTPART	y	y	y	y	y	y	y
CMPPART				y	y		
CPYPART	y	y	y	y	y	y	y
CRTPART			y	y	y		
CVTPART				y	y		
DLTPART	y	y	y	y	y	y	y
DSPPART	y	y	y	y	y		y
EXPPART	y	y	y	y	y	y	y
FNDSTRPART[2]				y	y		
IMPPART	y	y	y	y	y	y	y
MRGPART				y	y		
PRMPART[3]	y	y	y	y	y	y	y
PRTPARTINF	y	y	y	y	y	y	y
QRYPART	y	y	y	y	y	y	y
RNMPART	y	y	y	y	y	y	y
RTVPARTINF	y	y	y	y	y	y	y
WRKPARTPDM	y	y	y	y	y	y	y

Note:

The following items apply to the part types listed in the table:

1. Message file and include parts are not actually built, but can be referred to by parts you are building. The same is true for parts of type BNDDIR; however, they can be built if you specify SCOPE(*EXTENDED) on the BLDPART command. See “Setting the Build Scope” on page 148 for more information.
2. The FNDSTRPART command is valid for a system-supplied or user-defined type stored in a source file and for FILE(PF) parts.
3. In general, objects can be promoted if you specify EXTEND(*YES) on the PRMPART command.

Table 27. Application Development Manager/400 Types and How They Relate to CL Commands (Part 6 of 8)

Command	SCHIDX	JOBD	JOBQ	MSGQ	OUTQ	PARTL	PRDDFN	PRDLOD
BLDPART						y		
CHGPART	y	y	y	y	y	y	y	y
CHGPARTINF	y	y	y	y	y	y	y	y
CHKINPART	y	y	y	y	y	y	y	y
CHKOUTPART	y	y	y	y	y	y	y	y
CMPPART						y	y	y
CPYPART	y	y	y	y	y	y	y	y
CRTPART	y	y	y	y	y	y	y	y
CVTPART								
DLTPART	y	y	y	y	y	y	y	y
DSPPART	y	y	y	y	y	y	y	y
EXPPART	y	y	y	y	y	y	y	y
FNDSTRPART[1]							y	y
IMPPART	y	y	y	y	y	y	y	y
MRGPART							y	y
PRMPART[2]	y	y	y	y	y	y	y	y
PRTPARTINF	y	y	y	y	y	y	y	y
QRYPART	y	y	y	y	y	y	y	y
RNMPART	y	y	y	y	y	y	y	y
RTVPARTINF	y	y	y	y	y	y	y	y
WRKPARTPDM	y	y	y	y	y	y	y	y

Note:

The following items apply to the part types listed in the table:

1. The FNDSTRPART command is valid for a system-supplied or user-defined type stored in a source file and for FILE(PF) parts.
2. In general, objects can be promoted if you specify EXTEND(*YES) on the PRMPART command.

Table 28. Application Development Manager/400 Types and How They Relate to CL Commands (Part 7 of 8)

Command	SYSTEML	CBL36SRC	CBL36INC	DSPF36SRC	MNU36SRC	MSGF36SRC	OLC36SRC
BLDPART							
CHGPART	y	y	y	y	y	y	y
CHGPARTINF	y	y	y	y	y	y	y
CHKINPART	y	y	y	y	y	y	y
CHKOUTPART	y	y	y	y	y	y	y
CMPPART	y	y	y	y	y	y	y
CPYPART	y	y	y	y	y	y	y
CRTPART	y	y	y	y	y	y	y
CVTPART	y	y	y	y	y	y	y
DLTPART	y	y	y	y	y	y	y
DSPPART	y	y	y	y	y	y	y
EXPPART	y	y	y	y	y	y	y
FNDSTRPART	y						
IMPPART	y	y	y	y	y	y	y
MRGPART	y	y	y	y	y	y	y
PRMPART[1]	y	y	y	y	y	y	y
PRTPARTINF	y	y	y	y	y	y	y
QRYPART	y	y	y	y	y	y	y
RNMPART	y	y	y	y	y	y	y
RTVPARTINF	y	y	y	y	y	y	y
WRKPARTPDM	y	y	y	y	y	y	y

Note:

1. In general, objects can be promoted if you specify EXTEND(*YES) on the PRMPART command.

Table 29. Application Development Manager/400 Types and How They Relate to CL Commands (Part 8 of 8)

Command	RPG36SRC	RPG36INC	RPT36SRC	SRT36SRC	TXT36SRC	VRPGBIN	VRPGTXT
BLDPART							
CHGPART	y	y	y	y	y	y	y
CHGPARTINF	y	y	y	y	y	y	y
CHKINPART	y	y	y	y	y	y	y
CHKOUTPART	y	y	y	y	y	y	y
CMPPART	y	y	y	y	y	y	y
CPYPART	y	y	y	y	y	y	y
CRTPART	y	y	y	y	y	y	y
CVTPART	y	y	y	y	y	y	y
DLTPART	y	y	y	y	y	y	y
DSPPART	y	y	y	y	y	y	y
EXPPART	y	y	y	y	y	y	y
FNDSTRPART							
IMPPART	y	y	y	y	y	y	y
MRGPART	y	y	y	y	y	y	y
PRMPART[1]	y	y	y	y	y	y	y
PRTPARTINF	y	y	y	y	y	y	y
QRYPART	y	y	y	y	y	y	y
RNMPART	y	y	y	y	y	y	y
RTVPARTINF	y	y	y	y	y	y	y
WRKPARTPDM	y	y	y	y	y	y	y

Note:

1. In general, objects can be promoted if you specify EXTEND(*YES) on the PRMPART command.

Appendix C. Naming Rules

This section describes the rules you need to keep in mind when you create names for projects, groups, parts, and promote codes.

Project and Group Names

When you create a project or group with the CRTPRJ command (F6=Create on the Work with Projects display) or the CRTGRP command (F6=Create on the Work with Groups display), you provide two names for it. The 32-character name you specify on the PRJ or GRP parameter is the name that is used on all the project or group-related CL commands to identify this particular project. Some valid project names are PAYROLL, BIWEEKLYPAYROLL, PAY_ROLL_09.92, or PAY$ROLL. Some valid group names are VERSION1, V1PAY$ROLL, and SYS_TEST_VERSION2.

On the SHORTPRJ parameter, you specify the short project name that is used to create a unique AS/400 library name for groups in that project.

On the SHORTGRP parameter, you specify the short group name that is used to create a unique AS/400 library name for each group in the project.

The short project name and the short group name are combined to form the AS/400 library name. For example, if you have a project, PAYROLL with a short name of PAY and a group MASTER with a short name of MST, the AS/400 library name would be PAY.MST. If another group in the same project is created that is called TEST, with a short name of TST, its library name would be PAY.TST. You should not use a short project name that begins with the character Q since the system assumes such a library is a system library. The short project name can be up to 4 characters long. The short group name can be up to 5 characters long. The first character must be one of:

Alphabetic (A-Z)
Dollar sign ($)
Number sign (#)
At sign (@)

The remaining 3 characters for the short project name and remaining 4 characters for the short group name can be any combination of:

Alphabetic (A-Z) or numeric (0-9)
Dollar sign ($)
Number sign (#)
At sign (@)
Underscore (_)

The short project and short group names cannot contain:

Quotation mark (')
Period (.)
Embedded blank
DBCS characters or any other special characters

Some valid short project names are PR$, PAYR, or PR92. Some valid short group names are V1, V1PAY, or TSTV2.

Part Names

Each part is uniquely identified by a combination of four pieces of information:

Project The name of the project to which this part belongs (up to 32 characters).

Group The name of the group to which this part belongs (up to 32 characters).

Type The part type, for example, PGM, FILE, or RPGSRC (up to 10 characters). The AS/400 object types supported by the Application Development Manager/400 feature are listed in Table 5 on page 61.

Part A name that uniquely identifies the part for a specific project, group, and type (up to 10 characters).

The first character of the part name must be one of:

Alphabetic (A-Z)
Dollar sign ($)
Number sign (#)
At sign (@)

The remaining characters can be any combination of:

Alphabetic (A-Z) or numeric (0-9)
Dollar sign ($)
Number sign (#)
At sign (@)
Period (.)
Underscore (_)

The part name cannot contain:

Quotation mark
Embedded blank
DBCS characters or any other special characters

A fully qualified part name can thus be up to 84 characters long plus have separating blank spaces. An example of a fully qualified name would be PAYROLL DEVELOPER1 RPGSRC BIWKLY. The part is named BIWKLY; it has a type of RPGSRC; and it is located in group DEVELOPER1 in the project PAYROLL.

A part of a given name and type can only exist in a group that does not already contain a part of the same name and type. If a part of the same name and type already exists in the default search path of the group that is to contain the new part, a promote code of *NONE must be specified when the part is created, copied, or imported.

User-Defined Type Names

Each part is uniquely identified by a combination of four pieces of information:

Project The name of the project to which this part belongs (up to 32 characters).

Group The name of the group to which this part belongs (up to 32 characters).

Type The part type, for example, ZPGM, ZFILE, or ZRPGSRC (up to 10 characters). It is recommended that all user-defined part type names begin with Z.

Part A name that uniquely identifies the part for a specific project, group, and type (up to 10 characters).

The first character of the part name must be one of:

Alphabetic (A-Z)
Dollar sign ($)
Number sign (#)
At sign (@)

The remaining characters can be any combination of:

Alphabetic (A-Z) or numeric (0-9)
Dollar sign ($)
Number sign (#)
At sign (@)
Period (.)
Underscore (_)

The part name cannot contain:

Quotation mark
Embedded blank
DBCS characters or any other special characters

A fully qualified part name can thus be up to 84 characters long plus the separating blank spaces. An example of a fully qualified name would be PAYROLL DEVELOPER1 RPGSRC BIWKLY. This part is named BIWKLY; it has a type of RPGSRC; and it is located in group DEVELOPER1 in the project PAYROLL.

A part of a given name and type can only exist in a group that does not already contain a part of the same name and type. If a part of the same name and type already exists in the default search path of the group to contain the new part, a promote code of *NONE must be specified when the part is created, copied, or imported.

Promote Code Names

The naming restrictions for a promote code value are the same as those for the group name; that is, it can be up to 32 characters long and the first character must be one of:

Alphabetic (A-Z)
Dollar sign ($)
Number sign (#)
At sign (@)

The remaining 31 characters can be any combination of:

Alphabetic (A-Z) or numeric (0-9)
Dollar sign ($)
Number sign (#)
At sign (@)
Period (.)
Underscore (_)

Appendix D. Substitution Variables

The Application Development Manager/400 feature recognizes several character strings as substitution variables. This is similar to the way in which the Programming Development Manager utility provides substitutions for its user-defined options.

These substitution variables are recognized in a variety of ways:

1. **The Programming Development Manager utility**

   In the commands you use to work with PDM user-defined options. The substitution variables can be used from the following Programming Development Manager displays:

   Work with Projects using PDM
   Work with Groups using PDM
   Work with Parts using PDM
   Create User-defined Option

   The value returned for each variable depends on the display with which you are working. If any of these variables are used when working with libraries, objects, or members, the variable is replaced by *NULL.

2. **BLDOPT part**

   In compile commands you define in a build options part (part of type BLDOPT).

3. **CMD parameter**

   In the command string you define on the CMD parameter of the CHGADMACN command.

4. **BLDCMD parameter**

   In the command string you define on the BLDCMD parameter of the ADDADMLANG and CHGADMLANG commands.

Note: In those cases when a substitution variable is not applicable, the substitution variable will not be replaced. For example, the &ZJ substitution variable cannot be replaced with the name of the part being moved or copied, if the action you are defining on the CHGADMACN command is a delete action. Here, it would not make sense to use the substitution variable &ZJ. The substitution variable does not resolve if the required information is not available.

Table 30 (Page 1 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&A Object attribute	**PDM** If you are working with parts, &A is replaced by the AS/400 object attribute for a part. If you are working with projects or groups, &A is replaced by *NULL. **BLDOPT part** Not applicable **CMD parameter** Object attribute of the part being processed. **BLDCMD parameter** &A is replaced by *NULL.
&B List type	**PDM** If you are working with a list of projects, &B is replaced by R. If you are working with a list of groups, &B is replaced by G. If you are working with a list of parts, &B is replaced by P. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&C Option	**PDM** &C is replaced by the user-defined option code. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&D Last changed date	**PDM** If you are working with a list of parts, &D is replaced by the date the contents of the part was last changed. The value returned will be in the system format with separator characters. Otherwise, &D is replaced by *NULL. You must use this variable in single quotation marks (that is, '&D') since the date may contain a slash (/) which is used as an operator. **BLDOPT part** Not applicable **CMD parameter** Last changed date of the part being processed. **BLDCMD parameter** Last changed date of the part being compiled.

Table 30 (Page 2 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&E Run in batch	**PDM** &E is replaced by *YES if Y is specified in the *Run in batch* prompt on the Change Defaults display and *NO if N is specified. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&F Source file name	**PDM** If you are working with parts, &F is replaced by the name of the file that stores the part. This is valid only for the part types listed in Table 4 on page 59. **BLDOPT part** Source file name of the part being compiled. **CMD parameter** Source file name of the part being processed. When the *CPY action is specified on the ACTION parameter of the CHGADMACN command, this is the name of the *from* source file. **BLDCMD parameter** Source file name of the part being processed. &F is replaced by *NULL when a source file name is not available.
&G Job description library	**PDM** &G is replaced by the job description library value from the Change Defaults display. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&H Job description name	**PDM** &H is replaced by the job description value from the Change Defaults display. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&J Job description	**PDM** &J is replaced by the job description value from the Change Defaults display in the format library/job description. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable

Table 30 (Page 3 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&L Library name	**PDM** If you are working with projects, &L is replaced by *NULL. If you are working with groups, &L is replaced by the name of the AS/400 library associated with the group. If you are working with parts, &L is replaced by the name of the AS/400 library for the group that contains the part. **BLDOPT part** Library name associated with the group containing the part being compiled. **CMD parameter** Library name associated with the group containing the part being processed. For the *CPY action specified on the ACTION parameter of the CHGADMACN command, this is the name of the library from which the part is being copied. For example, if the IMPPART command is defined for the *CPY action, &L is replaced with the name of the library from which the part is being imported. When the EXPPART command is issued, this is the name of the library to which the part is being exported for the *DLT action specified on the ACTION parameter of the CHGADMACN command. **BLDCMD parameter** Library name associated with the group containing the part being compiled.
&N Object/member name	**PDM** If you are working with projects, &N is replaced by *NULL. If you are working with groups, &N is replaced by the name of the AS/400 library associated with the group. If you are working with parts, &N is replaced by the name of the AS/400 object or member associated with the item. **BLDOPT part** Object name of the part being compiled. **CMD parameter** Object name of the part being processed. **BLDCMD parameter** Object name of the part being compiled.
&O Object library	**PDM** If you are working with projects, &O is replaced by *NULL. If you are working with groups, &O is replaced by the name of the AS/400 library associated with the group. If you are working with parts, &O is replaced by the name of the AS/400 library associated with the group named in the *Specified group* prompt on the Work with Parts Using PDM display. **BLDOPT part** Name of the library where the output object of the BLDPART command is placed. **CMD parameter** Library name associated with the group containing the part being processed. For the *CPY action specified on the ACTION parameter of the CHGADMACN command, this is the name of the library from which the part is being copied. For example, if the IMPPART command is defined for the *CPY action, &O is replaced with the name of the library from which the part is being imported. When the DLTPART command is issued, this is the name of the library where the part resides. **BLDCMD parameter** Name of the library where the output object of the BLDPART command is placed.

Table 30 (Page 4 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&P Compile in batch	**PDM** &P is replaced with *YES if Y is specified in the *Compile in batch* prompt on the Change Defaults display and *NO if N is specified. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&R Replace object	**PDM** If you are working with projects, groups, or parts, &R is always replaced by *NO regardless of what is specified in the *Replace object* prompt on the Change Defaults display. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&S Item type without '*'	**PDM** If you are working with projects, &S is replaced by *NULL. If you are working with groups, &S is replaced by LIB. If you are working with parts, &S is replaced by the AS/400 object type of the part without the '*'. **BLDOPT part** Not applicable **CMD parameter** AS/400 object type of part being processed not preceded by '*'. **BLDCMD parameter** AS/400 object type of part being compiled not preceded by '*'.
&T Item type with '*'	**PDM** If you are working with projects, &T is replaced by *NULL. If you are working with groups, &T is replaced by *LIB. If you are working with parts, &T is replaced by the AS/400 object type of the part with the '*'. **BLDOPT part** Not applicable **CMD parameter** AS/400 object type of part being processed preceded by '*'. **BLDCMD parameter** AS/400 object type of part being compiled preceded by '*'.

Table 30 (Page 5 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&U User-defined option file	**PDM** &U is replaced by the user-defined option file name from the Change Defaults display. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&V User-defined option library	**PDM** &V is replaced by the user-defined option library name from the Change Defaults display. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&W User-defined option file member	**PDM** &W is replaced by the user-defined option file member name from the Change Defaults display. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&X Object text description	**PDM** If you are working with projects, &X is replaced by *NULL. If you are working with groups or parts, &X is replaced by the text (in single quotation marks), of the AS/400 object or member associated with the item. **BLDOPT part** Not applicable **CMD parameter** Object text description of the part being processed (50 characters). **BLDCMD parameter** Object text description of the part being compiled (50 characters).
&ZA Language	**PDM** &ZA is replaced by the part's language attribute. **BLDOPT part** Language attribute of the source part being compiled, not of the output part. **CMD parameter** Language of the part being processed. **BLDCMD parameter** Language of the part being compiled.

Table 30 (Page 6 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&ZC AS/400 command name	**PDM** &ZC is replaced by the name of the Application Development Manager/400 command being processed; for example, the IMPPART or BLDPART command. **BLDOPT part** &ZC is replaced with BLDPART. **CMD parameter** AS/400 command name currently being processed; for example, the IMPPART or BLDPART command. **BLDCMD parameter** &ZC is replaced with BLDPART.
&ZD Build output part type	**PDM** Not applicable **BLDOPT part** &ZD is replaced by the type of the output part being created by the BLDPART command. If the source part has not been previously built, this is the same as the type of the source part. **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&ZE Output part name	**PDM** Not applicable **BLDOPT part** &ZE is replaced by the name of the output part being created by the BLDPART command. If the source part has not been previously built, this is the same as the name of the source part. For a user-defined type of *NONE, &ZE is set to the first output member met by the BLDPART command, or the name of the part with the user-defined type of *NONE, if the part has not been processed before. **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&ZF Source file name	**PDM** Not applicable **BLDOPT part** Not applicable **CMD parameter** Library name associated with the group to which a part is being copied. You can use &ZF when defining *CPY action on the CHGADMACN command. **BLDCMD parameter** Not applicable

Table 30 (Page 7 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&ZG Group name	**PDM** &ZG is replaced by the name of the group given in the *Specified group* prompt on the Work with Parts Using PDM display. For the PRMPART command, this is the group from which the part is being promoted. For the CHKOUTPART command, this is the group to which the part is being checked out. **BLDOPT part** Name of the group where the outputs of the BLDPART command are placed. **CMD parameter** Name of the group containing the part being processed. **BLDCMD parameter** Name of the group containing the part being compiled.
&ZH Scan keyword	**PDM** &ZH is replaced by the value of the SCAN keyword from the Change Defaults display. The values are *YES and *NO. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&ZI Library name	**PDM** Not applicable **BLDOPT part** Not applicable **CMD parameter** Library name associated with the group to which a part is being moved or copied. You can use &ZI when defining the *MOV or *CPY action on the CHGADMACN command. **BLDCMD parameter** Not applicable
&ZJ TOPART value	**PDM** Not applicable **BLDOPT part** Not applicable **CMD parameter** Name of the part being moved or copied using a copy action. **BLDCMD parameter** Not applicable

Table 30 (Page 8 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&ZL Group name	**PDM** &ZL is replaced by the name of the group beside which the option was typed on the Work with Parts Using PDM display. If you are working with projects or groups, &ZL is replaced by *NULL. **BLDOPT part** Not applicable **CMD parameter** Not applicable **BLDCMD parameter** Not applicable
&ZN Name	**PDM** &ZN is replaced by the name from the Programming Development Manager list in use. If you are working with projects, &ZN is replaced by the project name. If you are working with groups, &ZN is replaced by the group name. If you are working with parts, &ZN is replaced by the part name. **BLDOPT part** Name of the part being compiled. **CMD parameter** Name of the part being processed. **BLDCMD parameter** Name of the part being compiled.
&ZO Build scope	**PDM** &ZO is replaced by the value of the Build Scope parameter on the Change Defaults display. **BLDOPT part** The value of the Build Scope parameter on the BLDPART command. The values are *NORMAL, *LIMITED, *DIRCHAIN, and *EXTENDED. **CMD parameter** Not applicable **BLDCMD parameter** The value specified in the command string on the BLDCMD parameter of the ADDADMLANG or CHGADMLANG commands. The values are *NORMAL, *LIMITED, *DIRCHAIN, and *EXTENDED.
&ZP Project name	**PDM** &ZP is replaced by the name of the project when you are working with groups or parts. **BLDOPT part** Name of the project containing the part being compiled. **CMD parameter** Name of the project containing the part being processed. **BLDCMD parameter** Name of the project containing the part being compiled.

Table 30 (Page 9 of 9). Substitution Variables Recognized by Application Development Manager/400

Substitution Variable	Description
&ZS Search path	**PDM** &ZS is replaced by the value of the SCHPTH keyword from the Change Defaults display. This value is the name of the search path part. **BLDOPT part** The value of the Search Path parameter on the BLDPART command. This value is the name of the search path part. **CMD parameter** Not applicable **BLDCMD parameter** The value specified in the command string on the BLDCMD parameter of the ADDADMLANG or CHGADMLANG commands.
&ZT Part type	**PDM** &ZT is replaced by the type of a part when you are working with a list of parts. **BLDOPT part** Type of part being compiled. **CMD parameter** Type of part being processed. **BLDCMD parameter** Type of part being compiled.
&ZX Text description	**PDM** &ZX is replaced by the text description (80 characters) of the item, depending on which display you are using. **BLDOPT part** Not applicable **CMD parameter** Text description of the part being processed (80 characters). **BLDCMD parameter** Text description of the part being processed (80 characters).
&ZY Part-list part name	**PDM** &ZY is replaced by the part-list part name used to generate the part-list on the Work With Parts in Part List display. It is *NONE on any other list in PDM. **BLDOPT part** Name of the part-list part specified on the PARTL parameter. **CMD parameter** Name of the part-list part specified on the PARTL parameter. **BLDCMD parameter** Name of the part-list part specified on the PARTL parameter.

Appendix E. Build Report Messages

This appendix lists the BLDPART command messages that appear on the build report to describe the reasons why a part is built. Both the first-level and second-level message text is provided.

Reason Messages

The following messages are the build report messages that describe the reasons why a part is built. In this section, &1=*type* and &2=*part*. *Type* and *part* resolve to actual names in the build report.

ADM4213 &1 &2 has changed.

Cause: The timestamp of the part &1 &2 has changed.

ADM4204 &1 &2 has no object.

Cause: The BLDPART command found the output part had no object associated with it. This was the result of a previous build process that removed the object to allow for compilation in the case of FILE PF and FILE LF.

ADM4203 &1 &2 not built before.

Cause: The BLDPART command found the output part with no information from a prior build process.

ADM4206 &1 &2 not created by source part.

Cause: The output part used to determine whether the build process should occur was previously built by another source part.

ADM4212 &1 &2 was not found.

Cause: This part was used in a previous build process, but could not be found in the build scope.

ADM4226 BLDPART processing cannot continue.

Cause: One or more work spaces that BLDPART uses internally have been exceeded. This can occur if a large number of parts is being built.

ADM4209 BLDOPT part contains compiler command.

Cause: The part was previously built using the default compiler command. The compiler command used in this build process came from a BLDOPT part.

ADM4211 BLDOPT part has changed.

Cause: The timestamp of the BLDOPT part has changed.

ADM4208 Default compiler command being used.

Cause: No BLDOPT part was found, and the default compiler command was used. The part was previously built using a BLDOPT part.

ADM4210 Different BLDOPT part found.

Cause: The BLDOPT part found for building this part was different from the one used in a previous build process.

ADM4216 Field changed in &1 &2.

Cause: A field in the specified FILE changed.

ADM4217 Field no longer found in &1 &2.

Cause: A field in the specified FILE no longer exists.

ADM4205 *FORCE set to *YES.

Cause: The FORCE parameter on the BLDPART command was set to *YES.

ADM4219 May be built because a dependency may be built.

Cause: The part did not require building, but a dependency may require building, which, in turn, may cause this part to require building.

ADM4221 May be built since source or dependency may build.

Cause: The part did not require building, but a dependency or the source that creates this part may require building which may cause this part to require building.

ADM4220 Must be built since source or dependency must build.

Cause: The part did not require building, but a dependency or the source that creates this part requires building which will cause the part to require building.

ADM4218 Must be built because a dependency must be built.

Cause: The part did not require building, but a dependency must be built, which causes this part to be built too.

ADM4202 Output part found was in a shared project.

Cause: The BLDPART command found the output part in a cross project. The build process continues as if there is no output part, meaning the source part will be built.

ADM4201 Output part from previous build process not found.

Cause: The BLDPART command could not find the output part created in a previous build process.

ADM4214 Record level id changed in &1 &2.

Cause: A record level id changed in the specified FILE.

ADM4215 Record no longer found in &1 &2.

Cause: A record in the specified FILE no longer exists.

ADM4207 Source part has changed.

Cause: The timestamp of the source part being built has changed since a previous build process was performed.

ADM4225 Source part has not been built before.

Cause: The source part has not been built before using the BLDPART command.

ADM4227 SRVPGM signature changed in &1 &2.

Cause: A signature changed in the specified SRVPGM.

ADM4228 User-defined language changed.

Cause: The user-defined language has been changed with the CHGADMLANG command.

Warning Messages

The substitution variables in the build report warning messages resolve to the following for most messages:

&1 Project name
&2 Group name
&3 Part type
&4 Part name

For some messages, &1, &2, &3, and &4 can represent library, object, and member names respectively. To see these messages, enter the following command.

```
DSPMSGD RANGE(ADM4400 *LAST) MSGF(QADM/QADMMSG)
```

Special Warning Messages

New warning messages will appear in the build report only when one of the internal build spaces exceed 80% of its capacity during the BLDPART processing. These warnings are also accompanied by a low level warning messages in the job log. Review these messages and perform appropriate recovery actions mentioned in the message descriptions of these messages.

Appendix F. Multi-Language Support

The Application Development Manager/400 feature gives you the ability to create and manage multiple versions of your application in more than one national language at a time, and to manage the translation activities required for the components that need translation.

This appendix provides a sample project hierarchy that you can create to manage a multi-language application, and explains what you need to consider when developing it. It does not discuss the national language support provided by IBM for this product. Refer to the *National Language Support* and *International Application Development* books for this information.

A Multi-Language Project Hierarchy

Assumptions

The scenario discussed in this section makes the following assumptions:

1. Development of your application is done in one base language by all your developers. (In this example, the base language is English.)
2. Some or all of the following types of application components must be translated:
   - Messages (parts of type MSGF).
   - Display files (parts of type DDSSRC).
   - Print files (parts of type PRTF).
   - Data areas (parts of type DTAARA).
   - Command source (parts of type CMDSRC).
   - Program source (parts of type xxxSRC or xxxINC, where xxx is one of the programming languages supported by this product. See Table 4 on page 59 for this information).

How the Project Hierarchy Works

Figure 88 on page 274 shows how the structure of a multi-language project hierarchy could be defined to help manage translation activities.

You create the project hierarchy as usual, taking into consideration any additional CCSID requirements. Refer to “Understanding the CCSID” on page 275 for an explanation of how the CCSID is used by Application Development Manager/400 commands.

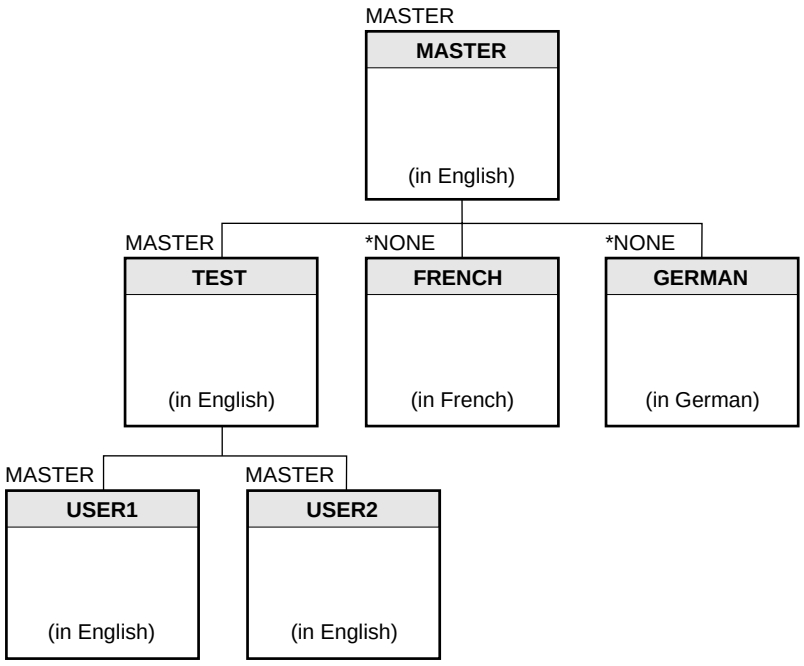

Figure 88. The Project Hierarchy for Translation

When a project is created, it takes on the double-byte character set (DBCS) characteristics of the system on which the project is created. This means that if the system is DBCS-enabled, the project is too, and its source files will allow DBCS characters to be used.

To develop the application in the base language, use the groups in the project hierarchy that have the same promote code as the root group. In this example, the promote code for the root group is MASTER. Use the groups with a promote code of *NONE to contain the copies of the parts being translated. In our example, the application is being translated into French and German.

Note: The groups FRENCH and GERMAN have been created with a promote code of *NONE so that parts from these groups cannot be promoted and replace the English version of the parts.

As the users develop the application, its parts are promoted, built, and tested up to the root group, MASTER. When all the parts of the application are in the root group, you can begin the translation of the language-sensitive parts:

1. Use the CHKOUTPART command to create a copy of the parts being translated in the group for the language you want to translate. Specify PRMCODE(*NONE) on these parts so that they can never be promoted.
2. After all the parts have been translated, build all the parts in the translation group, for example, the group FRENCH.

3. After all the parts in the group have been built, export the version of the application in the language required. For example, you can export your entire application from the root group if you want to create an English version, or you can export your entire application in French by specifying GRP(FRENCH) on the EXPPART command. If you prefer to export only the translated parts, do this by specifying the appropriate language's group name and SCAN(*NO) on the EXPPART command. For more information on the EXPPART command, refer to Chapter 13, “Exporting an Application.”

Understanding the CCSID

This section explains how the CCSID is used by the Application Development Manager/400 commands, and how the CCSID affects your translatable source. For a detailed discussion of the CCSID, you should refer to the *National Language Support* and *International Application Development* books.

The following Application Development Manager/400 commands use the CCSID:

- CRTGRP

  When you create a group, you specify, either explicitly or by default, the CCSID used to create all source files in that group. The CCSID only affects source members; it does not affect other objects directly.

  If a CCSID is relevant to the people translating the parts, you should create the group with that CCSID.[1]

- CPYPART, IMPPART, CHKOUTPART, PRMPART, and EXPPART

  When a source member is copied from one file to another using one of these commands, the hexadecimal values in the source member are translated from the copy-from CCSID representation to the copy-to CCSID representation. For the CPYPART, IMPPART, CHKOUTPART, and PRMPART commands, if the source file to receive the member does not exist, it is created with the CCSID of the group receiving the part. If the source file to receive the member does not exist for the EXPPART command, it is created with the same CCSID as the job running the EXPPART command.

The CCSID is only helpful when considering translatable source. However, it does not affect message files themselves because these cannot be created with a CCSID.

You can translate the hexadecimal values of parts in the base language group to those needed by other languages as described below. Refer to Figure 88 as we describe how translating the English parts to French and German is done:

- Messages

  To translate message files, check them out from the root group to the groups FRENCH and GERMAN using the CHKOUTPART command. You must then manually translate each file.

[1] It is recommended that the same CCSID be used for all the groups in the project hierarchy as those used in the development of the base language application. This prevents needlessly translating the hexadecimal representation of the source as it is promoted up the project hierarchy.

- Panel groups and UIM Menu

  The following three attributes are used with PNLGRP parts:

  – MSGID

    The compiler extracts the first-level message text for the message MSGID from the library MSGF in library LIB, and substitutes the text in place of this symbol. The message text is retrieved without any substitution variables.

    The message file defaults to the message file specified on the SUBMSGF attribute of the PNLGRP tag. If SUBMSGF is not specified, the message file must be specified in the symbol.

  – SUBMSGF

    This is the default message file used when a message file is not specified on the `&msg` symbol.

  – DFTMSGF

    This is the default message file used when a message file is not specified for a message identifier on any of the following tags: CHECK, LISTDEF, TL, DATASLTC, PDFLDC, LISTACT, and MENUI.

    If the corresponding message file attribute is not specified on any of these tags when the MSGID attribute is specified for this tag, this attribute is required.

- Display files

  There are three situations to consider for DDS source:

  – The DDS source contains MSGID keywords for all of its translatable text.[2]

    If this is the case, you need to translate only the messages in the method described above. This is because the messages and MSGID keywords are resolved at run time.

  – The DDS source contains MSGCON keywords for some or all of its translatable text.

    Translate the messages as described above. Then recompile the files in the groups FRENCH and GERMAN. You must do this because the messages and MSGCON keywords are resolved at compilation time. Compile the DDS source using the BLDPART command.

  – The DDS source contains literals for some or all its translatable text.

    To translate these parts, you must check them out from the root group to the groups FRENCH and GERMAN. Then, you must translate each part in the group manually. After the translation is complete, compile the DDS source using the BLDPART command.

- Data areas

  To translate data areas, first check them out from the root group to the FRENCH and GERMAN groups. Then translate each data area manually.

[2] Currently, the MSGID keyword is only supported in DSPF DDS source.

- Program source

  If you plan to translate your application into other languages, we recommend that you do *not* declare literals which you want to translate in your program source. However, if you have program source with this property, you can still translate it by doing the following. You must check the parts out from the root group to the groups FRENCH and GERMAN. Then, you must translate the literals in each part in the group manually. When the translation is complete, compile the program source using the BLDPART command.

- Command source

  To translate command source, first check the command source out from the root group to the groups FRENCH and GERMAN. Then translate each command source part manually. When the translation is complete, compile the command source using the BLDPART command.

Appendix G. Hints and Tips

This appendix contains suggestions for improving the efficiency and performance of some Application Development Manager/400 commands. This appendix provides information about the following topics:

- Performance improvements
- Part conversion: Importing a panel group source
- Setting up exit programs
- Miscellaneous hints and tips

Performance Improvements

This section provides information about the following topics:

- Improving WRKGRPPDM, WRKPARTPDM, and WRKPRJPDM displays, and part command performance
- Improving WRKPARTPDM and part command performance

Improving Performance

To improve performance of the part commands, and response time on the WRKGRPPDM, WRKPARTPDM and WRKPRJPDM displays, issue every night the following commands to reorganize the essential data base files used by Application Development Manager/400:

```
RGZPFM FILE(QUSRSYS/QALYPART) KEYFILE(*FILE)
RGZPFM FILE(QUSRSSYS/QALYBLDMAP) KEYFILE(*FILE)
RGZPFM FILE(QUSRSYS/QALYFLDTBL) KEYFILE(*FILE)
RGZPFM FILE(QUSRSYS/QALYREL) KEYFILE(*FILE)
```

Improving WRKPARTPDM and Part Command Performance

To improve the performance of the part commands, and response time on the WRKPARTPDM display, issue the following commands:

1. Find the size of the part directory file by typing:

   ```
   DSPOBJD OBJ(QUSRSYS/QALYPART) OBJTYPE(*FILE) DETAIL(*FULL)
   ```

2. Create subsystem:

   ```
   CRTSBSD SBSD(QGPL/ADMPARTS) POOLS((1 *BASE) (2 xxx 1))
   ```

   where xxx is greater than the size of the part directory file.

3. Start the subsystem:

   ```
   STRSBS (QGPL/ADMPARTS)
   ```

4. Clear the memory in the pool:

   ```
   CLRPOOL POOL(ADMPARTS 2) will clear the memory in the pool.
   ```

5. Set the object access:

   ```
   SETOBJACC OBJ(QUSRSYS/QALYPART) OBJTYPE(*FILE) POOL(ADMPARTS 2)
   ```

Part Conversion: Importing a Panel Group Source

To import a panel group source as a part of type PNLSRC with its language PNLGRP, follow these steps:

1. Change the member attribute of the member containing the panel group source to PNLGRP.
2. Import the member with LANG(*ATTR) specified on the IMPPART command.

This method is useful for any part type with multiple valid languages.

If you did not change the member attribute as desired before importing it, then you can use the CHGPARTINF command on the part created to convert its current language to the desired one. For example, if the attribute of the panel source member you imported was MENU, and you want to create a part of type PNLSRC with language PNLGRP, then you can change its language from MENU to PNLGRP using the CHGPARTINF command.

Setting Up Exit Programs For Additional Processing

Exit programs are programs that run additional processing when an Application Development Manager/400 command is processed. Because all Application Development Manager/400 functions are command driven, you can easily set up exit programs for any Application Development Manager/400 function that you want. Exit programs are flexible and can be used to insert additional logic before and after an Application Development Manager/400 command is processed.

Suppose that you want to add some extra processing to the DLTPART command. You can write an exit program that looks like the one shown in Figure 89.

```
PGM        PARM(&CMD &PRJ &GRP &TYPE &PART &PARTL +
                  &DLTARCHIVE)

DCL        VAR(&CMD) TYPE(*CHAR) LEN(2)
DCL        VAR(&PRJ) TYPE(*CHAR) LEN(32)
DCL        VAR(&GRP) TYPE(*CHAR) LEN(32)
DCL        VAR(&TYPE) TYPE(*CHAR) LEN(10)
DCL        VAR(&PART) TYPE(*CHAR) LEN(10)
DCL        VAR(&PARTL) TYPE(*CHAR) LEN(10)
DCL        VAR(&DLTARCHIVE) TYPE(*CHAR) LEN(4)

/* Insert Preprocessing Logic Here  */

QSYS/DLTPART PRJ(&PRJ) GRP(&GRP) TYPE(&TYPE) +
             PART(&PART) PARTL(&PARTL) +
             DLTARCHIVE(&DLTARCHIVE)

MONMSG     MSGID(ADM0000) EXEC(SNDPGMMSG MSGID(ADM9003)     +
             MSGF(QADM/QADMMSG))

/* Insert Postprocessing Logic Here */

ENDPGM
```

Figure 89. A Sample Exit Program for the DLTPART Command.

The above exit program does not yet perform any extra processing. You can, however, easily insert extra statements in the places indicated.

In case you need to obtain the native name of an Application Development Manager/400 part, you can use the Retrieve Part Information (RTVPARTINF) command in your exit program to retrieve this information. The RTVPARTINF command returns the library, file, and member values in separate parameters.

In order for this program to be called when a developer uses the DLTPART command, you will need to make a copy of the DLTPART command and place that copy in a library which is higher up in the library list than library QSYS. Your copy of the DLTPART command should be changed so that it is processed by your exit program.

The BLDPART command provides an even easier way to code an exit program without rewriting the command interface. First create a build-option part of type BLDOPT. You can call an exit program by specifying a label that is the same as the part type being processed in front of it. The labelled commands can be coded anywhere in a build-option part. If a labelled command is found in a build-option part, then this command rather than the compiler command for the source is processed, even though the compiler command is uncommented.

A sample of a BLDOPT QDFT part is shown in Figure 90.

```
        CRTRPGPGM ...
          /*******************************************************/
          /* PGM compilers                                       */
          /*******************************************************/
             CRTRPGPGM  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                          REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE)
          /* CRTCBLPGM  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                          REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE) */
             CRTCLPGM   PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                          USRPRF(*USER) REPLACE(*YES) AUT(*EXCLUDE)
                               .
                               .
                               .
CLPSRC:       CALL USRCRTCLP PARM(&O &ZE &L &F &ZN)

RPGSRC:       USRCRTRPG  PGM(&O/&ZE) SRCFILE(&L/&F) SRCMBR(&ZN) +
                          REPLACE(*YES) USRPRF(*USER) AUT(*EXCLUDE)
                               .
                               .
                               .
```

Figure 90. A Sample BLDOPT QDFT Part.

In the above example, the exit program command USRCRTRPG, rather than CRTRPGPGM, is invoked by the BLDPART command to process RPGSRC-type parts. The exit program command USRCRTRPG, a CL program that preprocesses the source, compiles the source by calling the CRTRPGPGM compiler, and finally postprocesses the output. The only requirement is that your exit program must call the appropriate compiler, since the compiler passes build-relationship information to the BLDPART command. You must remember the order in which BLDOPT parts are processed, and modify them as appropriate.

Miscellaneous Hints and Tips

This section provides information about the following:

- Using source files of different lengths

Using Source Files of Different Lengths

By default, Application Development Manager/400 creates source files with lengths of 92 bytes. (An exception is QRPGLESRC, which is created with a length of 112 bytes.) The source files are created when any part development command writes to it and does not find it. If the part development command finds a source file, then it adds, removes, or replaces the members in it.

If you use source file lengths that are different than the default length of 92 bytes, then create all source files in each group with the desired lengths as soon as the groups are created. You also need to create similar files in the archive group libraries. If the archive group libraries do not exist, then archive at least one of the source members stored in the 92-byte source files. Doing so will create an archive library.

Bibliography

This bibliography lists a variety of books that may be of use or interest to you as you work with the Application Development Manager/400 feature.

The Application Development Manager/400 library contains the following publications:

- *ADTS/400: Application Development Manager Self-Study Guide*, SC09-2138
- *ADTS/400: Application Development Manager API Reference*, SC09-2180
- *ADTS/400: Application Development Manager Introduction and Planning Guide*, GC09-1807
- *ADTS/400: Application Development Manager User's Guide*, SC09-2133

The Application Development ToolSet/400 library contains the following publications:

- *ADTS/400: Advanced Printer Function*, SC09-1766
- *ADTS/400: Character Generator Utility*, SC09-1769
- *ADTS/400: Data File Utility*, SC09-1773
- *ADTS/400: File Compare and Merge Utility*, SC09-1772
- *ADTS/400: Interactive Source Debugger*, SC09-1897
- *ADTS/400: Programming Development Manager*, SC09-1771
- *ADTS/400: Report Layout Utility*, SC09-1767
- *ADTS/400: Screen Design Aid*, SC09-1768
- *ADTS/400: Source Entry Utility*, SC09-1774
- *Introducing Application Development ToolSet for OS/400 and the AS/400 Server Access Programs*, GC09-2088

The Application Dictionary Services/400 library contains the following publications:

- *ADTS/400: Application Dictionary Services Self-Study Guide*, SC09-2086
- *ADTS/400: Application Dictionary Services User's Guide*, SC09-2087

The Application Development ToolSet Client Server/400 contains the following publications:

- *Installing Application Development ToolSet Client Server for OS/400*, SC09-2188
- *Introduction to Application Development ToolSet Client Server for OS/400*, GC09-2189

The VisualAge for RPG library contains the following publications:

- *VRPG Client for OS/2 Parts Reference*, SC09-1846
- *VRPG Client for OS/2 Language Reference*, SC09-1847
- *Client/Server Visual Programming with VRPG Client for OS/2*, SC09-2131
- *Getting Started with VRPG Client for OS/2*, SC09-2195

The CODE/400 library contains the following publications:

- *CODE/400 Debug Tool*, SC09-1905
- *Introduction to Application Development ToolSet Client Server for OS/400*, GC09-2189
- *CODE/400 Keyboard Template*, GX09-1297
- *CODE/400 Quick Reference Card*, GX09-1296
- *CODE/400 Self-Study Guide*, SC09-1911

The following publications in the AS/400 library may be of interest to you in relation to this feature:

- *Backup and Recovery – Basic*, SC41-4304
- *CL Programming*, SC41-4721
- *CL Reference*, SC41-4722
- *COBOL/400 User's Guide*, SC09-1812
- *COBOL/400 Reference*, SC09-1813
- *Data Management*, SC41-4710
- *DB2 for OS/400 Database Programming*, SC41-4701
- *DB2 for OS/400 SQL Programming*, SC41-4611
- *DB2 for OS/400 SQL Reference*, SC41-4612
- *DDS Reference*, SC41-4712
- *Experience RPG IV*, SC09-1938
- *ICF Programming*, SC41-3442
- *ILE Application Development Example*, SC41-3602
- *ILE Concepts*, SC41-4606
- *ILE C/400 Programmer's Guide*, SC09-2069
- *ILE C/400 Programmer's Reference*, SC09-2070
- *ILE C/400 Reference Summary*, SX09-1304
- *ILE COBOL/400 Programmer's Guide*, SC09-2072
- *ILE COBOL/400 Reference*, SC09-2073
- *ILE COBOL/400 Reference Summary*, SX09-1305

- *ILE RPG/400 Programmer's Guide*, SC09-2074
- *ILE RPG/400 Reference*, SC09-2077
- *ILE RPG Reference Summary*, SX09-1306
- *International Application Development*, SC41-4603
- *National Language Support*, SC41-4101
- *Publications Reference*, SC41-4003
- *Security – Reference*, SC41-4302
- *System API Programming*, SC41-4800
- *System API Reference*, SC41-4801
- *System Manager Use*, SC41-3321
- *System Operation for New Users*, SC41-3200

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W