IBM Application Program Driver/400 Version 3

SH12-6404-00

Developer's Guide

Release 6.0

IBM Application Program Driver/400 Version 3

SH12-6404-00

Developer's Guide

Release 6.0

Note!

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

First Edition, September 1995

This edition applies to Release 6 Modification Level 0 of IBM Application Program Driver/400 Version 3 (5716-PD1) and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. See the Summary of Changes for the changes made to this manual. Make sure you are using the correct edition for the level of the product.

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

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

IBM Anwendungssysteme GmbH
Information Development, Department 5160
Postfach 72 12 80
30532 Hannover
Germany

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

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent 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 Corporation, 208 Harbor Drive, Stamford, Connecticut 06904.

For online versions of this book, we authorize you to:

- Copy, modify, and print the documentation contained on the media, for use within your enterprise, provided you reproduce the copyright notice, all warning statements, and other required statements on each copy or partial copy.
- Transfer the original unaltered copy of the documentation when you transfer the related IBM product (which may be either machines you own, or programs, if the program's license terms permit a transfer). You must, at the same time, destroy all other copies of the documentation.

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization.

THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Your failure to comply with the terms above terminates this authorization. Upon termination, you must destroy your machine readable documentation.

Programming Interface Information

This book is intended to help developers enable AS/400 programs to run under APD/400, and develop programs that use the functions and services of APD/400.

This book also documents General-use Programming Interface and Associated Guidance Information provided by APD/400.

General-use programming interfaces allow the customer to write programs that obtain the services of APD/400.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

General-use programming interface

General-use Programming Interface and Associated Guidance Information...

End of General-use programming interface

Interfaces for Version 3 Release 6

Certain interfaces described in this book (the Audit file is an exception) are valid for Version 3 Release 6 of APD/400 and are not general-use programming interfaces. These interfaces may change in future releases of APD/400, and should not be used on target systems. The applicable interfaces are marked throughout this book.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

Application System/400	AS/400
Common User Access	CUA
Operating System/400	OS/400
IBM	OfficeVision/400
SQL/400	

About This Book

This book provides application developers with the necessary information to enable Application System/400* (AS/400*) application programs to run under IBM Application Program Driver/400 Version 3 (program number 5716-PD1), in the following referred to as APD/400, and to develop applications that use the functions and services of APD/400.

You can use the provided facilities and services for menu creation and control, authorization checking, exclusion control, installation support, and so on, instead of creating and maintaining these functions within your application.

Chapter 1, “Introduction” on page 1 describes the APD/400 concepts of installations, applications, and data sets, and how symbolic libraries and library name templates are used for applications that are multiinstallable or multi-data set enabled.

Chapter 2, “Developing an APD/400 Application” on page 6 explains how to develop an application to run under APD/400. Topics explain how to build and describe an application, including how to create an application interface program (AIP), how to modify an AIP, how to describe the application tasks and menus, authorization structure, exclusions, and menu Help texts, and how to display messages.

Chapter 3, “Packaging, Shipping, and Installing an APD/400 Application” on page 34 describes the procedures to develop an APD/400 application on a source system, package it, and install it on to a target system. Sample scenarios are included, and considerations are given for applications that are enabled for multilingual support.

Chapter 4, “User Exits and APIs” on page 50 contains a description of the user exits and application program interfaces (APIs) supplied with APD/400. For each user exit and API, a listing of the interface parameters is given.

Appendix A, “Layout of File QAAFTASK0” on page 101 shows the layout of the APD/400 Task file.

Appendix B, “Layout of File QAAFMENU0” on page 105 shows the layout of the APD/400 Menu file.

Appendix C, “Adding Tasks to the Task File” on page 107 shows how tasks can be added to the APD/400 Task file.

Appendix D, “Evaluating the APD/400 Audit File” on page 109 shows the layout of the APD/400 Audit file. Examples are included on how to create Audit file query reports.

The back of this book provides a glossary containing definitions of terms used across the APD/400 library, a bibliography, and an index.

Who Should Read This Book

This book is for application developers. Some knowledge and experience of the following is assumed:

- AS/400 computers and the Operating System/400* (OS/400*) operating environment.
- The administrative functions of APD/400 (see the *IBM Application Program Driver/400 Version 3: Administrator's Guide* for an explanation of these functions).
- AS/400 Control Language (CL) commands.
- How to create and modify a CL program using the Source Entry Utility (SEU) and the OS/400 command CRTCLPGM (create CL program).

Summary of Changes

The following major changes have been made since Version 3 Release 1:

- The API CHKEXC has been enhanced in two points.
  - It is possible to check/set exclusion for other than current data set.
  - If an exclusion is identified, information about this exclusion is provided via the API.

Chapter 1. Introduction

This chapter introduces APD/400 by describing how:

- APD/400 is divided into installations
- Applications can be installed into installations
- Application data is organized
- Symbolic libraries and library name templates are used for applications that are multiinstallable or multi-data set enabled.

Concept of Installations, Applications, and Data Sets

APD/400 is divided into installations. You could, for example, create different installations for testing, education, and production.

In the example in Figure 1, APD/400 is divided into the installations `bbb` (the default installation that always exists), `IN1`, and `IN2`.

Applications belong to installations; the same application can be installed more than once in different installations.

APD/400 itself can be considered as an application running under APD/400. For example, all APD/400 tasks (Administer, Select, and OfficeVision/400 functions) belong to the application `APD`. `APD` is installed only in installation `bbb` (3 blanks), although APD/400 tasks can be started from all installations. However, administering APD/400 tasks, authorization lists, and so on, is possible only from the installation `bbb`.

Application `APD+` is created automatically by APD/400 for every installation. The purpose of this application is to hold all personal menus and other user-created objects. `APD+` tasks are visible only in the installation to which they belong.

Data sets are used to separate different sets of data for one application, for example, data for different companies for a financial application. In the following example, application `AP1` installed in installation `IN1` has 3 different data sets: `DS1`, `DS2`, and `DS3`.

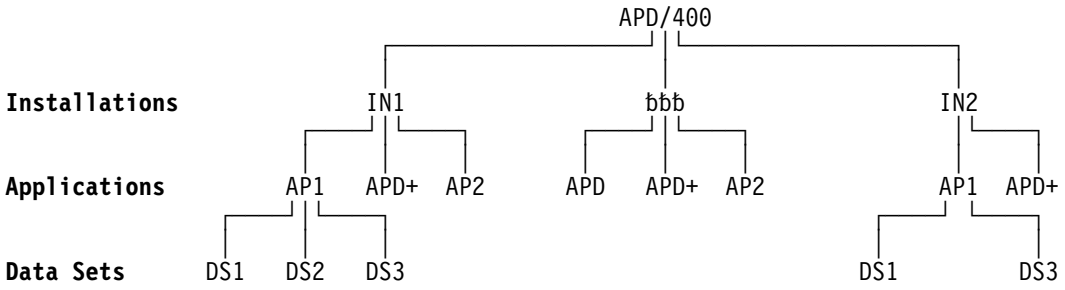

Figure 1. Installations, Applications, and Data Sets

Installations

The first subdivision is installations. An installation is identified by a 3-byte installation ID.

Every application running under APD/400 control belongs to an installation (which must be specified at installation time). The same application can be installed as many times as required if the application developer follows the rules described for symbolic libraries and library name templates on Pages 41 and 42. If the same application is installed twice (as, for example, `AP1` is installed in installations `IN1` and `IN2`), two copies of the complete application code must exist, that is, application objects and data, and APD/400 control objects (menus and authorization lists).

Installing the same application more than once can be used as follows:

- For having a production and a tutorial version of the same application.
- For a new release of an application for a customer. The new release is first installed into an installation other than the one containing the current production version. It is tested, and when it is found to be working correctly, it can become the production version.
- For a software developer who wants to have all versions of an application on the same system for maintenance reasons. This can be done by installing the application into different installations.

Every user has a "current" installation in which to work. This current installation can be changed using the `Select Installation` function (SLTINS).

Access to an installation can be secured by an authorization list. The authorization list used by APD/400 is called `INST_nnn`, where `nnn` is replaced by the installation ID. This authorization list belongs to the application `APD`, and can be changed using the `Administer Authorization Lists` function (ADMAUT).

Applications

The second subdivision is applications. Applications are identified by the installation ID and a 7-byte internal application ID. For IBM* applications, this is the product number; for non-IBM applications, a similar 7-byte identifier. For convenience, the customer may specify a so-called external application ID at installation time (it is easier to remember `APD` than `5763PD1`).

An unlimited number of applications may exist within one installation. The only limitations are that the applications must not share the same libraries, and that the internal and external application IDs must be unique within one installation.

All application definitions controlled by APD/400 (for example, menus and authorization lists), are identified by:

```
Installation ID +
(internal) Application ID +
Item name (for example, menu name).
```

A menu `MENU1` could exist twice within one installation if it belongs to different applications (for example, to `AP1` and to `AP2` in installation `IN1`).

Data Sets

The third subdivision is data sets. The concept of data sets allows you to have different sets of data for the same application with the same functional code and the same APD/400 application definitions.

Different technical methods to implement data sets are supported by APD/400:

- Separate libraries
- Separate members in database files
- Separate records in database files.

Using separate libraries is the default method supported by APD/400 at installation and runtime (see Page 42).

Methods using separate members and separate records in database files have runtime support but no installation support.

APD/400 supports selection of a data set with the `Select Data Sets` function (SLTDS). This function allows a user to select a data set from a list of data sets that the user is authorized to use. Authorization is provided by means of an authorization list that can be assigned to a data set using the `Administer Data Sets` function (ADMDS).

At runtime, APD/400 also supports data sets in the exclusion concept. Exclusions can be defined to be valid only within the same data set (for example, an application function is allowed to run with different data sets at the same time but is not allowed to run with the same data set at the same time), or for all data sets (for example, an application function is not allowed to run twice at the same time regardless of the data sets used).

Because the technical implementation is not limited, APD/400 does not support it at runtime. For example, setting library lists (using separate libraries), overriding database files to special members (using separate members), or working only with selected records in a database file (using separate records in database files), must be done in the application programs. The only support provided by APD/400 is that the names of the current installation, application, and data set are passed to the AIP.

Data sets can be added using:

- `Install Applications` (INSAPL)
- `Administer Data Sets` (ADMDS)
- The Install Application Definition API (INSAPPD)
- The Work with Data Sets API (WRKDST).

Symbolic Libraries and Library Name Templates

When an application is designed to be installed more than once under APD/400 (multiinstall enabled), and to work with data sets (multi-data set enabled), or both, the application developer has several problems:

- At installation time, the customer selects the name of the installation and data set. These names are not known when the developer packages the application.

- Multiinstall enabling (always) and multi-data set enabling (frequently) means that the names of application objects (libraries, members, and so on) can be determined only at install time.

  How does the developer instruct APD/400 to change OS/400 object names at install time?
- APD/400 supports Help for menus using display files or panel groups stored in libraries. If the name of the library is determined at installation time, how can the developer specify the name of the library at development time?
- APD/400's exclusion control provides allocation of OS/400 objects stored in libraries. If the name of the library is determined at installation time, how can the developer specify the name of the library at development time?

APD/400 provides a solution to these problems using library name templates and symbolic library names.

The library name template is used by APD/400 to form new library names based on the template plus the installation and data-set IDs selected by the customer at installation time. The following example might be the library name template:

Table 1. Example Library Name Template

Name on Tape	Symbolic Name	Library Name Template
OBJLIB	OBJLIB	OLB&I1.&I2.&I3.
DTALIB	DTALIB	DLB&D1.&D2.&D3.&D4.&I1.&I2.&I3.

The APD/400 `Install Applications` function restores library OBJLIB from tape to library OLBIN1 (for installation `IN1`), and to library OLBIN2 (for installation `IN2`). The data library DTALIB is restored to DLBIN1DS01 (for installation `IN1` and data set `DS01`), and respectively for the other data sets and installations. The symbolic name remains the same (OBJLIB and DTALIB), regardless of the resolved name.

The following table shows the list of libraries used in the APD/400 data repository after installation to `IN1` with data sets `DS01`, `DS02`, and `DS03`, and to installation `IN2` with data sets `DS01` and `DS03`.

Table 2. Libraries Used after Installation

Installation	Data Set	Symbolic Name	Resolved Name
IN1		OBJLIB	OLBIN1
IN1	DS01	DTALIB	DLBIN1DS01
IN1	DS02	DTALIB	DLBIN1DS02
IN1	DS03	DTALIB	DLBIN1DS03
IN2		OBJLIB	OLBIN2
IN2	DS01	DTALIB	DLBIN2DS01
IN2	DS03	DTALIB	DLBIN2DS03

You can now use the symbolic library name as follows:

- As the name of the library used to store online Help display files or panel groups (specified using the `Administer Applications (Developer)` function (ADMAPP).

If, for example, &OBJLIB (when the symbolic name is used it must be preceded by an ampersand) is specified for the Help library and the current installation is `IN1`, APD/400 searches library OLBIN1 for the Help object.

- As the name of the library used in an object exclusion list.

  For example, an object exclusion list is defined where the symbolic library name &DTALIB is specified as the library of a data area to be allocated *EXCL. The current installation is `IN1` and the current data set is `DS01`.

  If the exclusion is defined as type 1 (single data set), APD/400 would attempt to allocate the data area in library DLBIN1DS01.

  If the exclusion is defined as type 2 (all data sets), APD/400 would attempt to allocate the data area in libraries DLBIN1DS01, DLBIN1DS02, and DLBIN1DS03.

The main advantage of this method is that software developers are not concerned with library names on the target system. They specify the template to be used for an application, and use only symbolic library names in the online Help and exclusion definitions.

Chapter 2. Developing an APD/400 Application

This chapter details the procedures for developing an application to run under APD/400. The following are described:

- Building a New Application
- Using APD/400 Functions in Existing Applications
- Design Considerations
- Describing an Application
- Creating an AIP
- Modifying an AIP
- Describing Tasks and Menus
- Describing the Authorization Structure
- Describing Exclusions
- Describing Menu Help Texts
- Displaying Messages.

Building a New Application

The following are the steps to build a new application to run under APD/400:

Install APD/400

You must have APD/400 installed before you start. See the *IBM Application Program Driver/400 Version 3: Administrator's Guide* for details.

Define the New Application in APD/400

Every new application must be made known to APD/400. This can only be done by the person with the user profile of the APD/400 administrator.

The APD/400 administrator uses the `Administer Applications (Developer)` function (ADMAPP) to describe the new application to APD/400. This function can only be accessed using an expert code, and not via a menu. The APD/400 administrator should specify you (the application developer) as the administrator of the new application.

See “Describing the Application” on page 9 for details.

Plan the Menu Tree

After the APD/400 administrator has defined the application, you sign on with the user profile of the administrator of the new application. You are now ready to build your application.

Design the menu tree of your application. Consider the following:

- If your application needs protection by an authorization scheme, you can identify menus and tasks where authorization checking should be done. It saves time if you create the required authorization lists now, because you can then use the names of the authorization lists when you enter tasks and menus.

  For a description of authorization checking, see “Describing the Authorization Structure” on page 29.

- If you have programs that should not run at the same time, you can identify these programs.

  For a description of exclusion checking, see “Describing Exclusions” on page 30.

Describe the Menu Tree in APD/400

To describe the menu tree in APD/400, first enter the tasks (the leaves of the menu tree). The `Administer Menus` function described in *IBM Application Program Driver/400 Version 3: Administrator's Guide* is used to create tasks.

Once you have entered the tasks, you can build menus that contain these tasks, and higher-level menus that contain the defined menus. You can select to display menus in windows or full-screen, assign menu bars to menus, and pull-downs to menu bars. You can also enable text fields for multilingual support.

At this stage, you can also implement a naming scheme for the programs of your application.

You build the menu tree "bottom-up." This method gives you an early prototype for navigation through your new application.

Select Names for Libraries That Contain Programs and Data

The next step is to select names for the libraries in which you want to store the programs and data of your application.

Write the AIP

You must write a program to interface between APD/400 and your application programs. This program is the AIP.

Write the Programs of Your Application

Using APD/400 Functions in Existing Applications

Essentially, the procedure is the same as that described for building a new application. You do not have to plan the menu tree and program names, as they already exist.

If the existing application has stored a menu structure or task definitions in files, it may be possible to save development time. To do so, write a short program, or use SQL/400* statements to transfer data from the application's file to APD/400 files. See Appendix C, "Adding Tasks to the Task File" on page 107 for details.

Design Considerations

APD/400 integrates different applications. This means that an APD/400 administrator can define menus that contain tasks of different applications.

Each application has a different environment. APD/400 does not try to use parameters for the environment, but uses a more versatile method. The developer of an application writes an AIP, which is processed by APD/400 whenever a user selects a task of the application program.

The AIP sets the environment for the application, calls the application program, and resets the environment.

Using Libraries

When an APD/400 application is installed, a library QAPDIAHDR is created. This name is reserved for APD/400.

If your application can be installed multiple times, or if you want to take advantage of the “free” multiple data-set solution of APD/400, your programs cannot directly address libraries. To check this, inspect the outfile from the Display Program References (DSPPGMREF) command, to see the libraries used by your application. The field WHLNAM should contain only spaces, *LIBL, QTEMP, and so on. Hard-coded library names in this field mean that your application directly addresses libraries.

For multiple installability, there are limitations in naming your libraries. See the description of the `Library name template` field on page 42 for a description of how to build library names. Do not give your libraries names such as LIBRARY or DATALIB.

If you use three characters of the installation ID and four characters of the data-set ID as suffixes, you have only three characters remaining to uniquely name your libraries. Using only (the first) two characters of the data-set ID gives you a total of five fixed characters in your library names.

You are not limited in the maximum number of application libraries, but you must have at least one.

Using QTEMP

All applications share the same temporary library (QTEMP). Names like DTAARA or TEMP for objects in QTEMP should be avoided. Use unique names for the objects in QTEMP. You could use the following to ensure uniqueness:

```
#yyyyyyynn
```

Where:

```
yyyyyyy = the internal application ID
nn      = a serial number from 00 - 99
```

Also, your application may compete with itself in the case of multiple installations. You might try some naming algorithm combining the internal application ID and the installation ID. This is significant because of the limitation on the length of object names. A better solution could be to have a data area in QTEMP (for example, #yyyyyyy00) containing the previous (initialized) installation ID. If the installation changes, you can delete your objects from QTEMP and any overrides you have created, and reinitialize. Do not use commands such as CLRLIB LIB(QTEMP) or DLTOVR FILE(*ALL).

Do not use any names starting with Q for objects in QTEMP.

Using Folders

APD/400 cannot install subfolders, so you should only use a “root” folder for storing Help texts.

In the case of multiple installations of an application, there is only the last installed version of the APD/400-installed Help text available.

Because other applications may also install folders, avoid names like HELP or DOC. An acceptable naming convention would be to use the internal application ID as the folder name.

Restart

APD/400 accomplishes restart by passing the application number of the interrupted job (see field JOBNA in "Parameter Passed to the AIP" on page 14) with a flag signaling the restart (see field RSTKZ in "Parameter Passed to the AIP" on page 14) to the AIP.

In the case of a multiple screen application program, the program itself must establish the point at which the interruption occurred, and reroute to that point. This requires additional programming, for example, by writing routing information (such as display numbers) and data to an intermediate work file, but the result is that the user can be returned to the same display as when the interruption occurred.

Describing the Application

If you want to create a new application or use an existing application with APD/400, you must first describe the application. The application description can be created using the Administer Applications (Developer) function. To use this function, you need the APD_ADMIN authority.

Use the expert code (ADMAPP) to invoke the function. The following is displayed:

```
APD/ADMAPP          Administer Applications (Developer)


Type options, press Enter.
  2=Change   3=Copy to library QAPDIAHDR   12=Work with application libraries

Option     Application ID    Inst.   Application text
        External   Internal
  __    YOURAPPL   1234WP1           Text of your application
  __    OFC        5738WP1           AS/400 Office - OfficeVision/400









F3=Exit   F5=Refresh   F6=Add   F12=Cancel   F17=Position to
```

Figure 2. Administer Applications (Developer)

To add an application description to the APD/400 database, select F6 (Add). The first page of Add Application Entry is displayed. On this page, you enter:

`Application text`
A brief description of the application.

Notes:

1. This description is displayed on the `Administer Applications` menu and many of the APD/400 menus.
2. You can enter &msg symbols in this field to retrieve text for a specific language. For more information on multilingual support, see the *IBM Application Program Driver/400 Version 3: Administrator's Guide*.

`Application administrator`
The user ID (user profile) of the administrator responsible for this application.

Note: Administration functions for the new application can be performed only by this user.

If you ship your application, this user profile name is also shipped. If your application is installed on another driver, the application administrator has a user profile with this name. If a user profile with this name does not exist in the other driver, a user profile is created by APD/400 at installation time, with default parameters. On the installation tape, it should be the name of the owner of all application objects.

`Audit`
The default value for task auditing (`Y`=Yes, `N`=No).

When you have entered the required information, press F8 (Page Down) to display the second page of `Add Application Entry`. On this page, you enter:

`Display blank line if missing option(s)`
Enter `Y` to specify that a menu option that does not exist or the user is not authorized to use is represented by a blank line. Otherwise, the subsequent options and subheadings are moved up one line (see *IBM Application Program Driver/400 Version 3: Administrator's Guide* for more information on structuring menus).

If two or more sequential options are missing, only one blank line is displayed. No blank line is displayed if the missing option is immediately preceded by a column heading or subheading, or if it is the first option on a menu page or in a column.

Note: This is not valid for menus in windows.

`Menu heading name`
The name of the menu heading format to be used for the application (for more information on menu heading formats, see *IBM Application Program Driver/400 Version 3: Administrator's Guide*).

This value is used as the default value for the tasks of this application. It is overridden if a different value is specified for a specific task.

If a menu heading format name is not specified here or for the menu task, the menu is displayed with APD/400 default heading lines.

Note: Depending on environment parameters, no menu headings are shown in windows.

`Menu column format`
Specify whether you want to use single-column or double-column format for the application menus.

Note: Only single-column format can be used in windows.

`Menu bar`
The menu bar for this application that is used if no menu bar is described on the menu level.

Note: Menu bars are not used in windows.

`Display menus in windows`
The definition on application level of whether menus should be displayed in window or full-screen format. This is the default that APD/400 uses if nothing is specified for the tasks of the application.

On this page, you can also press F18 (Change attributes) to specify the presentation of full-screen menus and windows ("Administering Applications" in *IBM Application Program Driver/400 Version 3: Administrator's Guide* describes how to change display attributes).

When you have entered the required information, press F8 (Page Down) for the third page of `Add Application Entry`. On this page, you enter:

`Version`
The version number you want to give your application.

`Release`
The release number you want to give your application.

`Modification`
The modification number of your application, if applicable.

`Copyright information`
The copyright text for your application. The copyright information you enter is displayed in the message line whenever a user first enters your application.

When you have entered the required information, press F8 (Page Down) for the fourth page of `Add Application Entry` On this page, you enter:

`Help for menus`

`Type`
For your application menus, APD/400 can display Help information. You can provide this Help information as:

1. OfficeVision/400 documents
2. Display files
3. UIM panel groups
4. User exit.

OfficeVision/400 documents are stored in a folder, and display files and UIM panel groups are stored in a library.

In this field, enter the number representing the type of Help you want to use in your application. Enter `0` in this field if you do not want to use Help support.

`Folder/Library/User exit`
Input is required in this field if you have specified a value other than `0` in the `Type` field. Enter one of the following:

Folder name If you specified `1` (OfficeVision/400 documents).

Library name If you specified 2 (Display files) or 3 (UIM panel groups). Remember that ten characters is the maximum for a library name. You can use a symbolic library name here. A symbolic library name starts with an ampersand (&). See Page 41 for more information on symbolic libraries.

User-exit name If you specified 4 (User exit). Remember that ten characters is the maximum for a user-exit name.

If you have selected 1, 2, or 3 as the Help type, and specify *MRI as folder or library, APD/400 determines which language folder or library to use. That folder or library contains language-dependent data in whatever language the user selected for the application.

When you have entered the required information, press F8 (Page Down) to display the fifth page of Add Application Entry.

If you want APD/400 to call user exits when processing your application, on this page you specify the names of your user-exit programs.

Pre/post save/restore
: The name of the user exit to be called whenever a library belonging to the application is saved or restored. For more information, see “SAVRST Save/Restore” on page 61.

Administer data sets
: The name of the user exit to be called from Administer Data Sets whenever a data-set entry is created, modified, or deleted. For more information, see “ADMDSTE Administer Data-Set Entries” on page 53. If the field is left blank, no user exit is called.

When you have entered the required information, press F8 (Page Down) to display the sixth page of Add Application Entry. On this page, you enter:

Internal application ID
: Each application running under APD/400 must be uniquely defined by an internal application ID. If two applications are to run under the same installation ID, they must have different application IDs.

  For IBM applications, the IBM program number (without the hyphen) is the internal application ID.

External application ID
: The external ID of the application to be created.

  Note: This ID is generally presented to the user. It should, therefore, be short and meaningful.

Application interface
: The name of the AIP used by APD/400 to call the application. The AIP sets up the environment for the application.

Installation program
: The name of the Post-Installation user exit. (see “POSTINS Post-Installation” on page 60).

`Required application`
: The internal ID of an application required in relation to the current one. If you later create a tape from your application and the tape is installed on another driver, the APD/400 installation program checks that the required application is present on the other driver.

When you have entered the required information, press F8 (Page Down) to display the last page of `Add Application Entry`.

Availability of authorization checking, exclusion checking, multi-data-set-enabling, and working with APIs influences (reduces) performance in the interactive part of APD/400.

Depending on the customer environment, the administrator makes the decision whether to work, for example, without authorization checking within an application.

With this panel you can remove the availability of these functions to improve performance.

The decision as to whether the administrator works within APD without authorization checking is dependent on the customer environment.

These settings can be used:

- to customize a special environment
- to switch off functionality for test reasons while developing an application.

Switchable functions are:

- Use Authorisation Checking
- Use Exclusion Checking
- Work with Data Sets
- Work with Defaults in APIs.

Creating an AIP

APD/400 integrates applications. This allows menus with tasks from different applications to be created by the APD/400 administrator. A user can call a menu or program that runs under APD/400, and can switch from one application to another.

Different applications usually have different and incompatible library lists, job descriptions, and naming and calling conventions. They may have conflicting definitions of the local data area or of other common resources. Integrating applications must be done carefully.

To achieve integration, APD/400 allows developers to create an application environment using a program each time an application task is called from an APD/400 menu. This program is the AIP. An AIP is required for each program.

If you are the developer of an application and you want to write the AIP for it, you can use the sample programs, described later, as the basis on which to build an AIP. In some cases, you can take one of the sample AIPs, change the library names to the names of the libraries you use in your application, and use it directly.

Principles of the AIP

An application program is not called directly by APD/400; instead APD/400 calls the AIP of the application with information directing it to call the appropriate program. The AIP then creates the environment expected by the application program and performs the actual call.

The AIP is the link between APD/400 and tasks of the application. The AIP of an application must therefore be placed in the QUSRSYS library. When you develop an application, you must create the AIP in QUSRSYS.

The name of the program to be called is passed to the AIP as follows:

If PTYPE=X (a call to a user exit) and UEXPGM (user-exit program name) is not blank, call the program specified in UEXPGM. Otherwise, call the program specified in MPGMN.

Once you have created the AIP, use the CHGPGM command for the AIP program with USEADPAUT(*NO). Do not remove the observable information from the program.

When development of your application is complete and you ship it to a target system, the installation procedure copies the AIP you created for your application to the QUSRSYS library of the target system. To avoid naming conflicts, the installation procedure gives your AIP a new name.

Parameter Passed to the AIP

The following describes general-use programming interface and associated guidance information.

APD/400 passes all relevant information about the task to be performed to the AIP as a parameter. The parameter is a single character string. Your AIP must "unstring" the required information and ignore the remainder.

The parameter is described as a table of fields. Length, From, and To describe the position of the fields in the parameter. The names used for the fields are the ones used internally by APD/400. They are also used in the sample AIPs supplied with APD/400, and in the description of files QAAFTASK0 (see Appendix A, "Layout of File QAAFTASK0" on page 101) and QAAFMENU0 (see Appendix B, "Layout of File QAAFMENU0" on page 105).

Note: When PTYPE=X (user exit) or PTYPE=P (parameter entry program), there are restrictions. In these cases, certain information is not passed to the AIP. The APD/400 user exits are detailed in "User-Exit Descriptions" on page 53.

Table 3 (Page 1 of 2). Parameter Passed to the AIP

Field name	Length	From	To	Type	Description
INSID	3	1	3	A	Installation ID
INSTX	40	4	43	A	Installation description
ALIAS	7	44	50	A	Alias name of the application
ANWTX	40	51	90	A	Application description
FIRNR	4	91	94	A	Data-set ID
FIRNM	40	95	134	A	Data-set description

Table 3 (Page 2 of 2). Parameter Passed to the AIP

Field name	Length	From	To	Type	Description
USRID	10	135	144	A	User ID
USRTX	40	145	184	A	User description
MEPOS	2	185	186	A	Menu selection number
MTASK	10	187	196	A	Task ID
METTL	46	197	242	A	Task description
MPGMN	10	243	252	A	Program name
MPPRA	40	253	292	A	Program number (1-40)
MAURR	1	293	293	A	Control flag
RSTKZ	1	294	294	A	Restart flag
JOBNA	6	295	300	A	Old job number for restart
ERROR	1	301	301	A	Error flag
AUDTF	1	302	302	A	Audit flag
ANWID	7	303	309	A	Internal application ID
BRSTF	1	310	310	A	Authorization level
PTYPE	1	311	311	A	Program type
MPPRT	512	312	823	A	Program parameter
SRCLB	10	824	833	A	APD/400 source library
OBJLB	10	834	843	A	APD/400 object library
DTALB	10	844	853	A	APD/400 data library
JRNLB	10	854	863	A	APD/400 journal library
FRFLD	200	864	1063	A	Parameter extension
COLAT	34	1064	1097	A	Color attributes
LNGFC	4	1098	1101	A	Language feature code
JOBNR	6	1102	1107	A	Job number
CLCRQ	1	1108	1108	A	Close call required
CTYPE	1	1109	1109	A	Call type
LSTAP	7	1110	1116	A	Last application
NXTAP	7	1117	1123	A	Next application

The following are extended descriptions of the parameter fields:

INSID
Installation ID. The user selected this installation ID to run the application when the AIP was called.

INSTX
Installation description. The description of the installation ID specified in `INSID`.

ALIAS
Alias name of the application. This is the short name of your application as the user sees it. When the application was installed from tape, the APD/400 administrator specified this name. When you defined the application using the `Administer Applications (Developer)` function (ADMAPP), you specified this name.

ANWTX
Application description. This is the long name of your application as the user sees it.

FIRNR
Data-set ID. This field contains the data-set ID that the user selected. If your application does not support multiple data sets, this field is blank.

FIRNM
Data-set description. This field contains the text that describes the data-set ID. The `Administer Data Sets` function (ADMDS) can be used to enter the text.

USRID
User ID. Contains the name of the user profile with which the user has signed on to OS/400.

USRTX
User description. This field is defined by the APD/400 administrator using the `Administer User Entries` function (ADMUSR).

MEPOS
Number of the menu selection. This field is empty (spaces) if the task is selected by expert code.

MTASK
Name of the selected task. The expert code of the task for which the AIP is called.

METTL
Description of the selected task. This is the `Task text` field from the task definition.

MPGMN
Program name. This is the name of the program to be called. See also the description of the field `PTYPE`.

MPPRA
The first 40 characters of field `MPPRT` (program parameter).

MAURR
Control flag defined with the task.

RSTKZ
Restart flag:

0 This is a normal call for the task.

1 The application program was called in a restart situation. This is a restart of a previously abended call of the same task. The developer of the AIP specifies how this restart situation is handled in the application.

JOBNA
The old job number used for a restart (where RSTKZ=1). The restart concept of your application may have used the OS/400 job number of APD/400 as a key to store restart information. APD/400 stored its job number for restart and presents it in this field to the AIP.

ERROR
Error flag or return code. This field must be set in the AIP when control is returned from the application program. The convention of returning error conditions from an application program is specific to the application.

When the AIP is exited, APD/400 displays general messages to notify the user of the error situation.

An exception is the user exit for checking exclusions (see "CHKEXC Check Exclusion" on page 57). When the AIP is used to call this user exit, APD/400 receives a return code in this field. A return code of 1 means that the two exclusion lists involved exclude each other.

The return codes are:

0 No error.
1 An error has occurred.
2 A serious error has occurred.
3 An error has occurred that requires special attention. Depending on the restart flag specified for the task, APD/400 enters the task into the Work with Canceled Jobs list.

For process lists, a nonzero value causes the list to be interrupted. Subsequent tasks in the list are not processed.

AUDTF
Audit flag. Whether or not audit records are written for the task. This is the audit value that is currently active for the task.

ANWID
Current application ID of your application. This is the internal application ID, and not the ALIAS name of the application.

BRSTF
Authorization level. If the task is protected by an authorization list, APD/400 retrieves the authorization level for the current user from the authorization list. The task is called only if the authorization level is not zero.

If you require a more detailed authorization scheme in your application, create an interpretation of this field in your AIP or in all programs of your application.

If the task is not protected by an authorization list, APD/400 sets BRSTF to 1.

PTYPE
The program type as specified in field MPGMN. The value of this field can be:

I MPGMN contains the name of an interactive application program (EXTYP=I in the corresponding Task file).

B MPGMN contains the name of a batch application program (taken from MPGMN in the Task file when EXTYP=B).

P MPGMN contains the name of an interactive parameter entry program to get data for a batch application program (taken from PRPGM in the Task file when EXTYP=B).

X This is a call to a user exit (see “Calling User Exits from APD/400” on page 51).

MPPRT
The program parameter defined with the task.

When the AIP is used to call the user exit for checking exclusions, APD/400 passes the name of the exclusion list to be checked in this field.

SRCLB
Contains QAPD as the APD/400 source library.

OBJLB
Contains QAPD as the APD/400 object library.

DTALB
Contains QUSRSYS as the APD/400 data library.

JRNLB
Contains QUSRSYS as the APD/400 journal library.

FRFLD
Data structure for the APD/400 parameter extension (reserved if PTYPE is **not** P).

When PTYPE=X, see “Calling User Exits from APD/400” on page 51 for the layout of the FRFLD structure.

COLAT
The 34 bytes of this field are organized as follows:

- The first 30 bytes are used to specify the color of the following display areas. Three bytes are used for each area.

  Menu bar
  Expert code
  Title
  Installation/Data set
  Top instruction
  Option area
  More/Bottom
  Command line
  Function keys
  Window border.

  Colors are represented in the three bytes as follows:

  000 Green
  001 Turquoise
  010 White
  011 Pink
  100 Blue
  101 Yellow
  110 Red.

- Byte 31 is used for reverse image on window borders:

  1 Set reverse image attribute on.
  0 Set reverse image attribute off.

- Byte 32 is used for the blink attribute on window borders:

  1 Set blink attribute on (use blink only with color red).
  0 Do not set blink attribute on.

- Bytes 33 and 34 are used to select a window border style:

  01 Common User Access*-1 (CUA*-1) (Style 1).
  00 Asterisks (Style 2).
  10 CUA-2 (Style 3).
  11 Blanks (Style 4).

The following example is part of a DDS record definition. The indicators 88 - 94 are used to control the style, color, and attribute of the window:

```
A          R WINDOW01
A                                      WINDOW(7 6 15 61)
A N88N89N90                            WDWBORDER((*COLOR GRN))
A N88N89 90                            WDWBORDER((*COLOR TRQ))
A N88 89N90                            WDWBORDER((*COLOR WHT))
A N88 89 90                            WDWBORDER((*COLOR PNK))
A  88N89N90                            WDWBORDER((*COLOR BLU))
A  88N89 90                            WDWBORDER((*COLOR YLW))
A  88 89N90                            WDWBORDER((*COLOR RED))
A*
A  91                                  WDWBORDER((*DSPATR RI))
A  92                                  WDWBORDER((*DSPATR BL))
A*
A N93N94                               WDWBORDER((*CHAR '...:::.:'))
A N93 94                               WDWBORDER((*CHAR '********'))
A  93N94                               WDWBORDER((*CHAR '.-.II''-'''))
A  93 94                               WDWBORDER((*CHAR '        '))
```

The following code could be used to transfer the color attributes from the AIP parameter to the indicator area passed to the workstation function manager:

```
CHGVAR %SST(&indara 88 7) %SST(&colatr 28 7)
```

LNGFC

The language feature code. For example, 2929 for German, or 2931 for Spanish. See *AS/400 National Language Support Planning Guide* for more information.

JOBNR

The job number of the current session.

CLCRQ

The application must return the information before performing a task of another application, whether a close call is required or not.

Example

1. Task TASK1 of application APPL1 was performed.
2. Return from this call, CLCRQ=1.
3. Task TASKX of application APPLX was required
4. APD recognises a change in the appplication ID (field ANWID), and performs a close call because CLCRQ=1. This close call is a normal AIP call with the interface contents of the previous call and the information that it is

a close call (CTYPE=3). The AIP has to recognise that CTYPE=3 and has to perform a special close handling.

5. After performing the close call, the required call task TASKX of application APPLX will be performed.

CTYPE

The call type tells the AIP whether:

1. an application is called the first time (CTYPE=1)
2. an application is called repeatedly (CTYPE=2)
3. the current call is a close call (CTYPE=3).

LSTAP

If the application ID has changed since the previous call, the application ID of the previous call is filed in LSTAP.

NXTAP

If the application ID has changed since the previous call, and a close call was required from the previous application (CLCRQ=1), a close call will be performed before performing the current call. In this close call, the information which is the application of the next call is included.

Sample AIP

Three sample AIPs are supplied with APD/400. To use one of these sample AIPs for your application, copy the appropriate sample from the APD/400 source file and modify as required. Select a name for your AIP that is unique within your QUSRSYS library.

The following sample shows a minimum AIP. The source code for this sample is in member SMPAIP01 in the APD/400 sample source files QAPD/QAAFSMPL0 and QUSRSYS/QAAFSMPL0.

```
/*-------------------------------------------------------------------*/
/*                           DISCLAIMER                              */
/*-------------------------------------------------------------------*/
/* Code is shown for illustrative purposes only. IBM has not fully   */
/* tested the code. THE CODE IS OFFERED "AS IS," AND ALL WARRANTIES  */
/* EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE       */
/* IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR PARTICULAR  */
/* PURPOSES, ARE EXPRESSLY DISCLAIMED. IBM is not liable for         */
/* any damages, including but not limited to consequential,          */
/* incidental, special or indirect damages from this code.           */
/*-------------------------------------------------------------------*/
/*                                                        SMPAIP01   */
/*  Minimum Application Interface Program                            */
/*                                                                   */
/*  Only interactive programs are called.                            */
/*  This program must be placed in the library QUSRSYS.              */
/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/* Called by APD/400 menu control                                     */
/*-------------------------------------------------------------------*/
SMPAIP01: +
PGM &parm

/*-------------------------------------------------------------------*/
/* Definition of variables                                           */
/*-------------------------------------------------------------------*/
DCL &parm       *CHAR 1200 /* Entry and exit parameters for the AIP.  */
DCL &mpgmn      *CHAR   10 /* Name of the program to be called.       */

/*-------------------------------------------------------------------*/
/* Unpack entry parameter                                            */
/*-------------------------------------------------------------------*/
CHGVAR  &mpgmn %SST(&parm 243 010)

/*-------------------------------------------------------------------*/
/* Set the library list for your application.                        */
/* Replace YOURLIBL by your library list before compiling this       */
/* program !                                                         */
/*-------------------------------------------------------------------*/
CHGLIBL LIBL(YOURLIBL QTEMP QGPL)

/*-------------------------------------------------------------------*/
/* Call the interactive program.                                     */
/*-------------------------------------------------------------------*/
CALL &mpgmn

/*-------------------------------------------------------------------*/
/* End program and return to APD/400                                  */
/*-------------------------------------------------------------------*/
RETURN:
ENDPGM
```

Additional Sample AIPs

The two other AIPs supplied with APD/400 are a full-function AIP and a standard AIP.

Full-Function AIP

This sample AIP:

- Determines the name of the program to be called.
- Determines the installation, application, and data-set IDs. This information must be retrieved from the AIP parameter for task processing and user-exit calls.
- Changes the library list based on the selected installation, data set, and language.
- Interprets the general purpose `Control flag` as a command or program as follows:

  0 Command
  1 Program.
- Transfers the AIP parameters to the local data area. Because the purpose of this application is to demonstrate APD/400, many functions use or change the AIP parameters; this is easiest done using the parameters in the local data area.
- Calls the application or user-exit program, or processes the application command.
- Passes all messages received from the operating system or the application to APD/400 for possible display. If the severity code of the message is greater than or equal to `20` (and the message ID is not equal to `CPF9898`), the `Error flag` is set to `1`.
- Transfers the local data area back to the AIP parameters.

The source code for this program is in member SMPAIP02 in the APD/400 sample source files QAPD/QAAFSMPL0 and QUSRSYS/QAAFSMPL0.

Standard AIP

This sample AIP interfaces APD/400 to many standard applications. It provides three possibilities for processing:

1. A command is processed.

   In this case, the field `Program Name` contains `CMD`, and `Program Parameter` contains the command (possibly including prompt characters).
2. A program is called.
   In this case, the field `Program Name` contains the name of the program, and `Program Parameter` contains any call parameters. The field `Control Flag` must contain `N`.
3. A System/36 procedure is processed.

   In this case, the field `Program Name` contains the name of the procedure, and `Program Parameter` contains any call parameters. The field `Control Flag` must contain `Y`.

The library list is “remembered” in a data area in the APD/400 library named by the combination of installation ID and internal application ID. Any blanks in the installation ID are replaced by hash symbols (#). The installation ID must **not** begin with a digit. If the library list has not yet been “remembered,” the program prompts the

CHGLIBL command and saves the input. If the list is changed subsequently, it is not saved.

The source code for this program is in member SMPAIP03 in the APD/400 sample source files QAPD/QAAFSMPL0 and QUSRSYS/QAAFSMPL0.

Modifying an AIP

The following are functions that you can include in an AIP to meet the requirements of an application:

- Changing the library list
- Allowing for multiple installations
- Allowing for multiple data sets
- Allowing for multilingual support
- Changing the job description
- Starting commitment control
- Saving the local data area
- Opening files
- Putting a time lock on your application.

Changing the Library List

Use the Change Library command (CHGLIBL) in your AIP to use a library list that is different from the default list. For example:

```
CHGLIBL LIBL(QTEMP ABC2 QGPL) CURLIB(ABC1)
```

Allowing for Multiple Installations

To allow for multiple installations of your application, use statements such as the following:

```
DCL      VAR(&ABC1) TYPE(*CHAR) LEN(10)
DCL      VAR(&ABC2) TYPE(*CHAR) LEN(10)
-------------------------------------
CHGVAR   VAR(&ABC1) VALUE('ABC1' *CAT &INSID)
CHGVAR   VAR(&ABC2) VALUE('ABC2' *CAT &INSID)
CHGLIBL  LIBL(QTEMP &ABC2 QGPL) CURLIB(&ABC1)
```

Note: Your application must not directly address libraries (see “Using Libraries” on page 8).

Allowing for Multiple Data Sets

To allow for multiple data sets as well as multiple installations of your application, use statements such as the following (assuming that ABC2 is the data library, and ABC1 contains only nonobjects):

```
CHGVAR VAR(&ABC1) VALUE('ABC1' *CAT &INSID)
CHGVAR VAR(&ABC2) VALUE('ABC2' *CAT &INSID +
  *TCAT %SST(FIRNR 1 2)) /* Chars 1-2 of data sets  */
CHGLIBL LIBL(QTEMP &ABC2 QGPL) CURLIB(&ABC1)
```

Allowing for Multilingual Support

To use different libraries (depending on the language) for textual data, use statements such as the following:

```
LNGFC 2929
CHGVAR VAR(&OBJLIB)
CHGVAR VAR(&OBJLIB) VALUE('ABC1' *CAT &INSID)
CHGVAR VAR(&MRILIB) VALUE('ABC2' *CAT INSID +
  *TCAT %SST(&LNGFC 3 2)
CHGLIBL LIBL(QTEMP &MRILIB &OBJLIB QGPL)
```

For more information on the APD/400 multilingual support feature, see *IBM Application Program Driver/400 Version 3: Administrator's Guide*.

Changing the Job Description

If, for example, you want to place the spool files of a payroll application into a protected queue, add the following statements before calling the application program:

```
DCL      VAR(&OUTQ) TYPE(*CHAR) LEN(10)
DCL      VAR(&OUTQLIB) TYPE(*CHAR) LEN(10)
----------------------------------------
RTVJOBA OUTQ(&OUTQ) OUTQLIB(&OUTQLIB)
  /* Save current out queue and library */
CHGJOB  OUTQ(ABCOUTQ)
```

After returning from the application program, add the statement:

```
CHGJOB  OUTQ(&OUTQLIB/&OUTQ)
```

Note: This illustrates the principle that when you make changes to the environment for your application, you must restore the original environment before returning to APD/400.

Starting Commitment Control

If some of your application programs run under commitment control, and the control flag (MAURR) can be used to indicate commitment control, you can insert the following statement before the program call:

```
IF COND(&MAURR *EQ '1') +
  THEN(STRCMTCTL LCKLVL(*CHG))
```

After the program call (restore the environment to end commitment control), insert:

```
IF COND(&MAURR *EQ '1') +
  THEN(ENDCMTCTL)
```

Note: If you have already used the control flag for some other purpose, you can use, for example, character 1 of the program parameter as the commitment control flag.

Saving the Local Data Area

Concerning the local data area, there are special considerations for applications running under APD/400:

- The local data is shared by all programs running in a particular job
- Several applications can run in the same job under APD/400
- Each application can have its own standards for what is put into the local data area.

These considerations are only relevant if your application uses the local data area to pass data from your current menu control program to a menu selection (application program), or from one menu selection to another.

If you use the local data area to pass data to subroutines or from a menu selection to a batch job, these considerations are not relevant.

If these considerations are relevant, your AIP must create and restore the local data area before calling the application program and, where necessary, save it before returning to APD/400 (otherwise the next task called may destroy it).

You can use the following statements to create and restore the local data area before calling the application program:

```
DCL        VAR(&LDA) TYPE(*CHAR) LEN(10)
---------------------------------------
RTVDTAARA  DTAARA(QTEMP/#1111TST01 (RTNVAR(&LDA)
  /* Get the contents of the previously saved local data area */
MONMSG     MSGID(CPF1015) +
  EXEC(DO)
    / If this is the first time ... */
    CHGVAR    VAR(&LDA) VALUE('Start value for LDA')
      /* ... create the initial contents ...*/
      /* (this is only an example */
    CRTDTAARA  DTAARA(QTEMP/#1111TST01) Type(*CHAR) +
      LEN(1024) VALUE(&LDA) TEXT('Contents of LDA')
      /* ... and create a data area to save it */
  ENDDO
CHGDTAARA  DTAARA(*LDA) VALUE(&LDA)
  /* Restore the local data area with previous/initial contents */
```

Before returning to APD/400, you can save the local data area using the following statement:

```
CHGDTAARA  DTAARA(QTEMP/#1111TST01) +
             VALUE(%SST(*LDA 1 1024))
  /* Save the new local data area contents in a data area */
```

Note: For a restart, this example may not work correctly. It is therefore better to save the contents of the local data area in a file, using the job number as key. To do this, write a high-level language (HLL) program to call the file from your AIP.

Opening Files

There are certain functions that you may require only once for each job, such as creating temporary files or preopening certain files to improve the performance of your application. You could use the following program before calling an application program to open files for the applicable installation:

```
DCL        VAR(&OLDINS) TYPE(*CHAR) LEN(3)
-----------------------------------------
RTVDTAARA  DTAARA(QTEMP/#1111TST00) RTNVAR(&OLDINS)
             /* Get the last installation ID */

MONMSG     MSGID(CPF1015) +
  EXEC(DO)
    /* If there was none ... */
    RMVMSG     CLEAR(*ALL)
                 /* ... delete system message ... */
    CHGVAR     VAR(&OLDINS) VALUE('...')
                 /* Initialize to impossible value */
    CRTDTAARA  DTAARA(QTEMP/#1111TST00) Type(*CHAR) +
                 LEN(3) VALUE(&OLDINS)
                 /* ... and create the data area */
ENDDO

IF         COND(&OLDINS *NE '...'   +
           *AND &OLDINS *NE &INSID) +
    THEN(DO)
      /* If the installation has changed */
      CLOF        OPNID(ABC1)
                   /* ... close the old files */
      CLOF        OPNID(ABC2)
    ENDDO

IF         COND(&INSID *NE &OLDINS) +
  THEN(DO)
    /* If the installation is new or changed ... */
    OPNDBF     FILE(ABC1) OPTION(*ALL) TYPE(*PERM)
                 /* ... open the files ... */
    OPNDBF     FILE(ABC2) OPTION(*ALL) TYPE(*PERM)
    CHGDTAARA  DTAARA(QTEMP/#1111TST00) VALUE(&INSID)
                 /* ... and save the installation ID */
  ENDDO
```

Note: If you have multiple data sets stored in different libraries or members, and you want to use the APD/400 `Select Data Sets` function (SLTDS) to switch between them, you may need similar logic to change between data sets.

Putting a Time Lock on the Application

You may have certain requirements that must be performed before a program in the application is called. For example, you may want to:

- Allow the use of the application only during normal working hours
- Restrict the use of the application
- Display a warning if the libraries are not saved
- Insert a record in a file for your internal restart.

Because APD/400 menu control does not call an application program directly, the AIP becomes a prolog for each of your programs. So, if you want to perform a task before a program is called, write it into the AIP.

As an example, you could add the following code to ensure that the application is used only between certain times of the day:

```
DCL        VAR(&TIMELOCK) TYPE(*CHAR) LEN(9)
DCL        VAR(&QTIME) TYPE(*CHAR) LEN(4)
-----------------------------------------
RTVDTAARA  DTAARA(TIMELOCK) RTNVAR(&TIMELOCK)
             /* Get the allowed time  from - to  */
             /* The format is        "HHMM-HHMM"  */
RTVSYSVAL  SYSVAL(QTIME) RTNVAR(&QTIME)
             /* Get the system time (HHMM)       */

IF COND(%SST(&QTIME 1 4) *LT %SST(&TIMELOCK 1 4)   +
   *OR  %SST(&QTIME 1 4) *GT %SST(&TIMELOCK 6 4)   +
  THEN(DO)
    /* If outside of the allowed time ... */
    SNDPGMMSG  MSGID(ABC1111) MSGF(ABCMSGF) +
                 TOPGMQ(*SAME QAFDRMAIN)
                 /* ... send an error message ... */
    CHGVAR     VAR(&ERROR) VALUE('1')
                 /* ... set the error flag ... */
    GOTO       CMDLBL(RETURN)
                 /* ... and return to APD/400. */
 ENDDO
```

Note: You do not have to write a program to allow for the modification of this data area; instead you can use the OS/400 selective prompting function in an APD/400 task description. See the *AS/400 Programming: Control Language Programmer's Guide* for information on selective prompting.

You could provide the following task to the user:

Task:	TIMES
Task type:	P
Processing type:	I
Menu option text:	Change application time frame
Authorization:	(Administrator only)
Program name:	PROG1
Program parameter:	`CHGDTAARA TIMELOCK ??VALUE('HHMM-HHMM')`

Using this technique, the user does not need to know the name of the data area, or the name of the command.

Set and Reset an Application Environment

APD/400 recognises whether an application is called for the first time, or repeatedly.

To improve performance, it is possible to open the application individual environment with the first task call of an application and close this environment with the last task call of the same application.

You can use the following statements to handle this requirement:

```
DCL     VAR(&CLCRQ) TYPE(*CHAR) LEN(1)
DCL     VAR(&CTYPE) TYPE(*CHAR) LEN(1)
DCL     VAR(&LSTAP) TYPE(*CHAR) LEN(7)
DCL     VAR(&NXTAP) TYPE(*CHAR) LEN(7)
-------------------------------------
CHGVAR  VAR(&CTYPE) VALUE(%SST(&PARM 1109 1)
CHGVAR  VAR(&LSTAP) VALUE(%SST(&PARM 1110 7)
CHGVAR  VAR(&NXTAP) VALUE(%SST(&PARM 1117 7)
CHGVAR  VAR(&CLCRQ) VALUE('1')

/* If it is an initial call, open the environment */

IF COND(CTYPE *EQ '1') +
  THEN(DO)

/* do everything that would normally be done for each task */
/* of that application, eg., opening files and so on.      */
/* It is also possible to do this, depending on the        */
/* previous application &LSTAP.                            */

END DO

/* If it is a close call, close the environment. */

IF COND(&CTYPE *EQ '3') +
  THEN(DO)

/* cancel everything which was done for an initial call */
/* of that application, eg., close files and so on.     */
/* It is also possible to make this, depending on the   */
/* next application &NXTAP.                             */

END DO

/* After opening the environment for an initial call */
/* and for a repeated call, the usual task call can  */
/* be performed.                                     */

IF COND(&CTYPE) *NE '3'
  THEN(DO)

/* perform usual task call */

END DO

/* APD needs the return information from the application */
/* whether a close call for the application is requried  */
/* or not                                               */

CHGVAR(%SST(&PARM 1108 1)) VALUE(&CLCRQ)
```

Describing Tasks and Menus

You describe tasks and menus for your application using the `Administer Menus` function (ADMMNU). This function is described in *IBM Application Program Driver/400 Version 3: Administrator's Guide*.

To create your menus, you must work "bottom up," describing the tasks for the menu selections before describing the menu structure:

1. Create task descriptions for all tasks in your application.
2. Create menus using these tasks.
3. Create menus on a higher level, using the tasks and menus already created.

Notes:

1. The menus need not be strictly hierarchical; you can leave some branches or some tasks (for example, commonly used commands) unattached to the main menu tree.
2. Take care when using type C (command) tasks. These tasks are called from APD/400 directly, and not through the AIP of your application. Application programs called as commands do not therefore find the APD/400 environment created in the AIP.
3. Tasks referring to a program named SIGNOFF are intercepted by APD/400 that, depending on the circumstances, processes a command such as ENDGRPJOB or SIGNOFF.

If you want to create APD/400 menus to access tasks of an existing application that has task information and a menu structure stored in files, you can save time by writing a conversion program that stores task descriptions and a menu tree directly in the corresponding APD/400 files QAAFTASK0 and QAAFMENU0. For an example of such a program, see Appendix C, "Adding Tasks to the Task File" on page 107. QAAFTASK0 is described in Appendix A, "Layout of File QAAFTASK0" on page 101, and QAAFMENU0 in Appendix B, "Layout of File QAAFMENU0" on page 105.

Describing the Authorization Structure

To describe the authorization of your application, use the `Administer Authorization Lists` function described in *IBM Application Program Driver/400 Version 3: Administrator's Guide*.

There are several points to consider:

- You should define authorization lists first and then define your tasks and menus. If you define tasks and menus before authorization lists, you must use the `Administer Menus` function again to enter the authorization list name for existing tasks and menus.
- You should generally protect all tasks, even SIGNOFF or DSPMSG, by mapping them to an authority list. In this way, the application administrator is able to make the complete application "invisible" to unauthorized users.
- Avoid defining too many authorization lists, to reduce the workload for the application administrator in authorizing users. If the administrator needs more flexi-

bility (more authorization lists) than you supply as “default,” new definitions can be easily created.

- The structure of authority often parallels menu structures. For example, assume that you have one menu with selections for administering master data, a second for entering various transactions, a third for different types of processing, and a fourth for printouts. It would be reasonable for you to define five authority lists: one protecting each of your menus and all underlying tasks, and a fifth protecting the main menu and whatever general utility functions you provide.
- In the authority file on your installation tape, leave the *ALL entry set to `0` to lock out users not specifically authorized, and include one record for each list authorizing the application administrator.

As the application developer, you must inform the administrator about the authorization structure of the application, enabling the administrator to assign the users to the correct authorizations.

Note: You can switch off authorization checking in the function `Administer Applications`

API for Authorization

It may be necessary to check the authorization of a user within an application program. For example, to check whether the user is allowed to:

- Cancel an order in the order entry program
- Post accounts for a certain account number or account group
- View or change employee salaries.

In your application program, you can call an APD/400 API to find the authorization level of a user in a given authorization list. See “CHKAUT Check Authorization” on page 73.

Note: If authorization checking is switched off, it has no influence on API CHKAUT.

User Exit for Authorization

For each menu item, APD/400 can determine the authorization level of a user from the authorization list specified with the task. If the authorization level is greater than `0`, APD/400 displays the menu item. If the authorization level is `0`, APD/400 does not display the item.

If the authorization level of a user within an authorization list cannot be determined in this way, you can provide a user exit with the authorization list. The authorization level is determined in this user exit. See “CHKAUT Check Authorization” on page 56.

Describing Exclusions

To describe the exclusions in your application, use the `Administer Exclusions` function as described in *IBM Application Program Driver/400 Version 3: Administrator's Guide*. Consider the following points:

- Reduce the number of conflicts that need to be defined by grouping your program tasks according to data usage. For example, you could put all pro-

grams that display or print a customer file into a single exclusion list, you could make two exclusion lists (displays and printouts), or you might want to put all programs that display or print master files into a single exclusion list.

- Program (and command) tasks that have no explicit mapping to an exclusion list are checked against a possible *ALL exclusion.
- You may have exclusions that are not easily (or not at all) definable with the syntax of APD/400. Sequential exclusions are of this type. In this case, you can write a user exit that informs APD/400 if an exclusion situation exists. Although this type of situation may already be coded into your application, it is good practice to use the APD/400 user exit. Otherwise, you lose certain advantages APD/400 offers, such as automatic rescheduling of batch jobs, maintainability by the customer, and same function and appearance as other applications.

Note: If exclusion checking is switched off for an application, then no exclusion checking will be performed for the tasks of exclusion lists of this application.

User Exit for Exclusions

You can use the APD/400 `Administer Exclusions` function to specify that a user exit determines your application exclusions. See the *IBM Application Program Driver/400 Version 3: Administrator's Guide* for a description of this function.

Whenever a user selects a task to process, APD/400 checks if the task is contained in any exclusion list. If so, and if that exclusion list is an exclusion of type 3 (user exit), APD/400 calls the user exit specified.

If the return code from the user exit indicates that the task is excluded, APD/400 does not start the task.

See “CHKEXC Check Exclusion” on page 57 for the technical specification of the user-exit interface.

API for Exclusions

An API is provided to set, reset, and check for existing exclusions in your application. This API is described in “CHKEXC Check Exclusion” on page 74.

Note: If exclusion checking is switched off, this has no influence on API CHKEXC.

Describing Menu Help Texts

There are four methods of providing Help texts for the menus in your application. You can use:

- OfficeVision/400 folders
- Display files
- UIM panel groups
- User exits.

Help Texts in Folders

OfficeVision/400 is used to create Help texts in folders. When you specify the Help method for a task in the `Administer Menus` function (ADMMNU), you specify a document and label. Consider the following when using this method:

- All menu Help texts for an application must be in a single folder. All installations of the same application must share the same folder. This folder must not be a subfolder.
- All Help texts must be in final format.
- You can use one large document containing all Help texts for the complete application, separate documents for each menu, program, and so on, or a combination of these methods.

Help Texts in Display Files

If you create display files to provide Help in your application, you must specify the following keywords in the display definition:

- INDARA on the file level
- LVLCHK(*NO) on the Create Display command.

If Help is requested by a user, one record from the display file is displayed.

Note: It may confuse the user if you have a record smaller than a full screen.

When you specify the Help method for a task in the `Administer Menus` function (ADMMNU), you specify a display file and record.

Help Texts in Panel Groups

To provide Help in panel groups, you create the panel groups using the Create Panel Group command in UIM. The maximum length of the name parameter on the HELP tag is 10 characters. For more information, see *AS/400 Guide to Programming Application and Help Displays*.

When you specify the Help method for a task in the `Administer Menus` function (ADMMNU), you specify a panel group and Help module.

Help Texts Using User Exits

To provide Help texts using user exits, you specify identifiers for the Help method for a task in the `Administer Menus` function (ADMMNU). The identifiers allow the user exit to determine which Help to display.

See “DSPHLP Display Help” on page 58 for a description of the user exit that enables an application to display Help texts to users.

Displaying Messages

Your application programs or the AIP can send messages to APD/400. For interactive jobs, messages are shown in the message line of the menu displayed after returning from your program. The message line is scrollable, if more than one message is in the queue. For batch jobs, messages are sent to the job log.

There are two methods of sending messages from an application program to APD/400:

- Use the Send Message API (see “SNDMSG Send Message” on page 92).
- Send the messages to the program that calls the AIP.

Notes:

1. In earlier releases of APD/400, a method using the ADPD010 program message queue was documented to send messages from an application program to APD/400. APD/400 Version 3 Release 1 does not support this program message queue.

   Change your AIP and any other programs that send messages to the ADPD010 program message queue to use one of the methods listed earlier.

2. In earlier releases of APD/400, a method using the ADMSGF0 message file was documented to send a status message ('Loading program. Please wait...'). APD/400 Version 3 Release 1 does not support the ADMSGF0 message file.

   Change your AIP and any other programs that use the ADF0042 message to use a different message file, or to use a hardcoded message as in the following example:

```
CHGVAR     &msgdta  'Loading Program. Please wait...'
SNDPGMMSG  MSGID(CPF9898) MSGF(QCPFMSG) MSGDTA(&msgdta) +
           TOPGMQ(*EXT) MSGTYPE(*STATUS)
```

Chapter 3. Packaging, Shipping, and Installing an APD/400 Application

Once you have completed development of your application on a source system (for example, as a software developer or in the head office), you need to package the application and ship it to a target system (for example, to a customer or a branch office) on which APD/400 is installed.

This chapter explains how to do this. The following subjects are covered:

- "Overview" describes the methods of packaging, shipping, and installing an application running under APD/400:
  - The standard APD/400 method, using the `Install Applications` function, automatically installs the APD/400 (menus, authorizations, and so on) and OS/400 parts (programs and database files) of an application. This method can be used only for the base installation of an application.
  - An individual procedure that uses APD/400 APIs to extract, compare, install, change, and delete the APD/400 part of the application. The OS/400 part of the application is packaged, shipped, and installed using OS/400 commands (for example, the SAVxxx and RSTxxx commands) or SystemManager/400 (SM/400). This method can be used for the base installation and the update of an application.
- "Creating the Standard Product Package" on page 39 describes how you can create a product package to be installed using the APD/400 standard method.
- "Sample Scenarios" on page 45 describes sample scenarios that explain how base versions and updates of an application can be packaged, shipped, and installed using APD/400.
- "Considerations for Multilingual Support" on page 49 lists considerations for applications that are enabled for multilingual support.

The sources for all programs described in this chapter are supplied with APD/400 in both the QAPD/QAAFSMPL0 and QUSRSYS/QAAFSMPL0 source files. Any changes you make to these sources will be overwritten the next time you install or restore APD/400. If you want to change the sources, copy the files to another library and make the changes there.

Overview

Basically, there are two different situations in which you send application code to a target system:

- The application is currently not installed on the target system. The complete application must be packaged, shipped, and installed. This is referred to as the "base installation" of an application in this chapter.
- The application is already installed on the target system. Only the changes between the previous version (installed on the target system) and the current version (installed on the source system) must be packaged, shipped, and installed. This situation is referred to as an "update" to an application in this chapter.

The following terms are used in this chapter:

Product package
: This is a set of libraries and folders that make up an application or an application update.

Shipment
: This is the act of transferring a product package from a source system to a target system. This can be done using physical media (tape, cartridge, diskette, or CD-ROM) or electronically (a savefile send with the SNDNETF command).

Installation
: When the product package is received on the target system, the installation process reads the data from the package and stores it permanently on the target system.

Application definitions and the AIP are referred to as the "APD/400 part" of an application in this chapter.

Application definition
: In the APD/400 data repository, the following application definitions are stored:[1]

  - Application description
  - Menus, tasks, menu bars, and so on
  - Menu heading formats
  - Authorization lists
  - User groups
  - Exclusions and exclusion lists
  - Batch environments
  - Timetables
  - Data-set entries
  - Application library descriptions.

  Note: The physical representation of application definitions in the APD/400 data repository may change in future releases of APD/400. However, APD/400 will automatically migrate the data from earlier versions to the current physical representation. The earliest version currently supported is Version 1 Release 1.

  Because migration is a slow process, it is strongly recommended that you create new product packages (based on the latest version of APD/400) when a new version of APD/400 is installed on the target system.

AIP
: The AIP is part of the application definition. However, it is treated separately in this chapter because it is a program and the other application definitions are records in a database file.

Application libraries and folders are referred to as the "OS/400 part" of an application in this chapter.

[1] This data is stored in OS/400 database files.

Application libraries

Libraries contain the objects that make up your application. However, some of the objects stored in application libraries are related to APD/400. For example:

- Display files or panel groups for online Help
- User-exit programs.

Application folders

Folders can contain document library objects (DLOs). These can be documents, or, for example, PC files (if your application is running on PCs and the AS/400). OfficeVision/400 documents for online Help in your application are also stored in folders.

Base Installation of an Application

The following scenario describes how you can package, ship, and install an application designed to run under control of APD/400.

On the source system, the current version of an application is installed. The objective is to build a package of this version, ship it to the target system, and install it on the target system. Figure 3 shows this procedure:

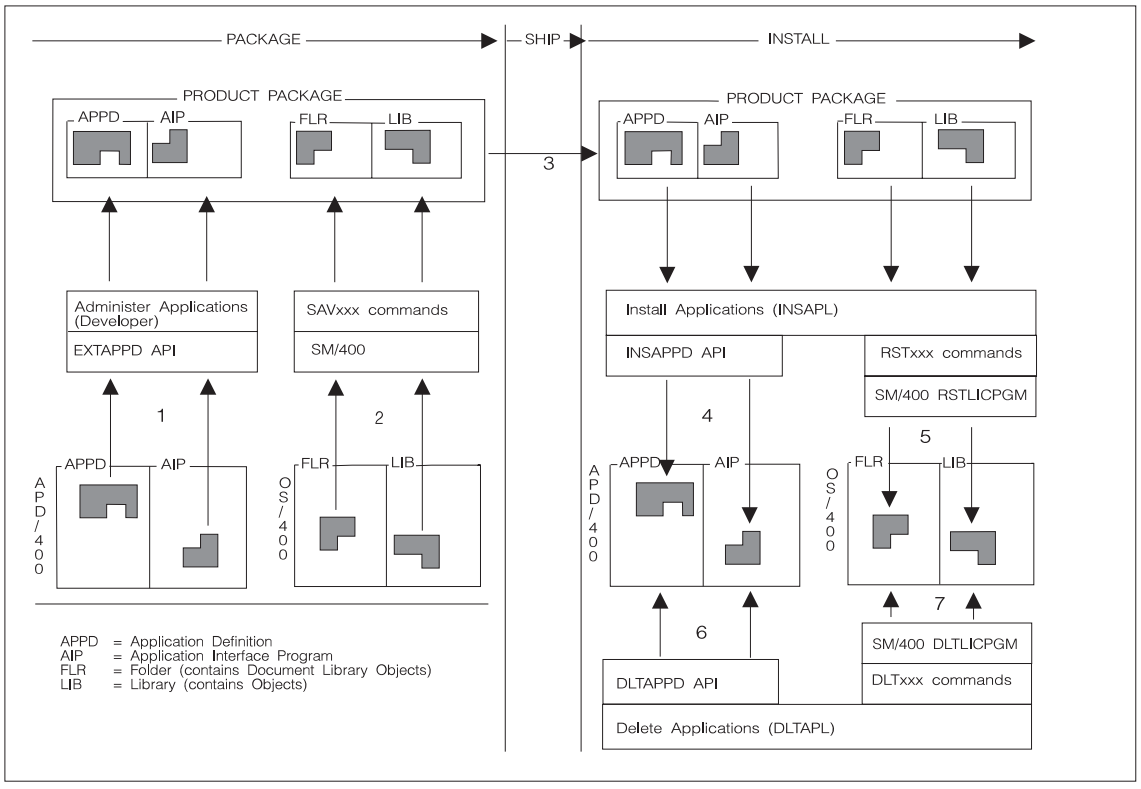

Figure 3. Package, Ship, and Install the Base Installation

1 The application definitions and the AIP of the current version are extracted from the APD/400 data repository using one of the following methods:

- Option 3 (Copy to library QAPDIAHDR) of the Administer Applications (Developer) function (ADMAPP). This function uses the library name QAPDIAHDR for the library that receives the application definitions.
- The Extract Application Definitions API (EXTAPPD).

2 The OS/400 libraries and folders are packaged. This is done using the OS/400 save commands (for example, SAVLIB or SAVDLO), or SystemManager/400 functions. "Creating the Standard Product Package" on page 39 gives useful tips and techniques to help you do this.

3 The product package is shipped to the target system, either physically (tape, cartridge, diskette, or CD-ROM) or electronically (as a savefile).

4 The APD/400 part of the application is installed using either the Install Applications function (the standard method), or the Install Application Definitions API (INSAPPD) for individual install procedures.

5 The OS/400 part of the application is installed using either the Install Applications function (standard method), OS/400 commands (for example, RSTLIB or RSTDLO), or SystemManager/400 (RSTLICPGM).

Although this scenario deals with the installation of an application, the APD/400 methods to deinstall or delete applications are mentioned here for completeness:

6 The APD/400 part of the application is deleted using either the Delete Applications function (standard method), or the Delete Application Definitions API (DLTAPPD) for individual install procedures.

7 The OS/400 part of the application is deleted using either the Delete Applications function (standard method), OS/400 commands (for example, DLTLIB or DLTDLO), or SystemManager/400 (DLTLICPGM).

Updating an Application

The following are considerations in updating an application that runs under APD/400 control:

- The new or changed APD/400 parts of the application (that is, new menus) are updated. For example:
  - Adding new tasks and menus because of functional enhancements
  - Changing conflict control information
  - Modifying the authorization scheme of your application
  - New timetables, changed batch environments, and so on.

  All such changes are retrieved from the source system by APD/400 APIs. These APIs extract the changes between the two versions of your application that can be shipped to a target system. Other APIs can be used on the target system to install the changes. Changes that have been applied previously on the target system (for example, new menus or changed authorizations) are preserved as far as possible. For example, if a menu has been modified on the target system but has not been modified between the previous and the current version, the changes on the target system are not affected.
- The new or changed OS/400 objects and DLOs are updated.
  The changes to the OS/400 part of an application can be applied using the OS/400 RSTxxx commands or SystemManager/400.

The following scenario describes this process.

On the source system, the previous (PRV) and the current (CUR) versions of an application are installed. On the target system, only the previous (PRV) version is installed. The previous version on the target system is upgraded with the current version from the source system. Figure 4 shows the procedure:

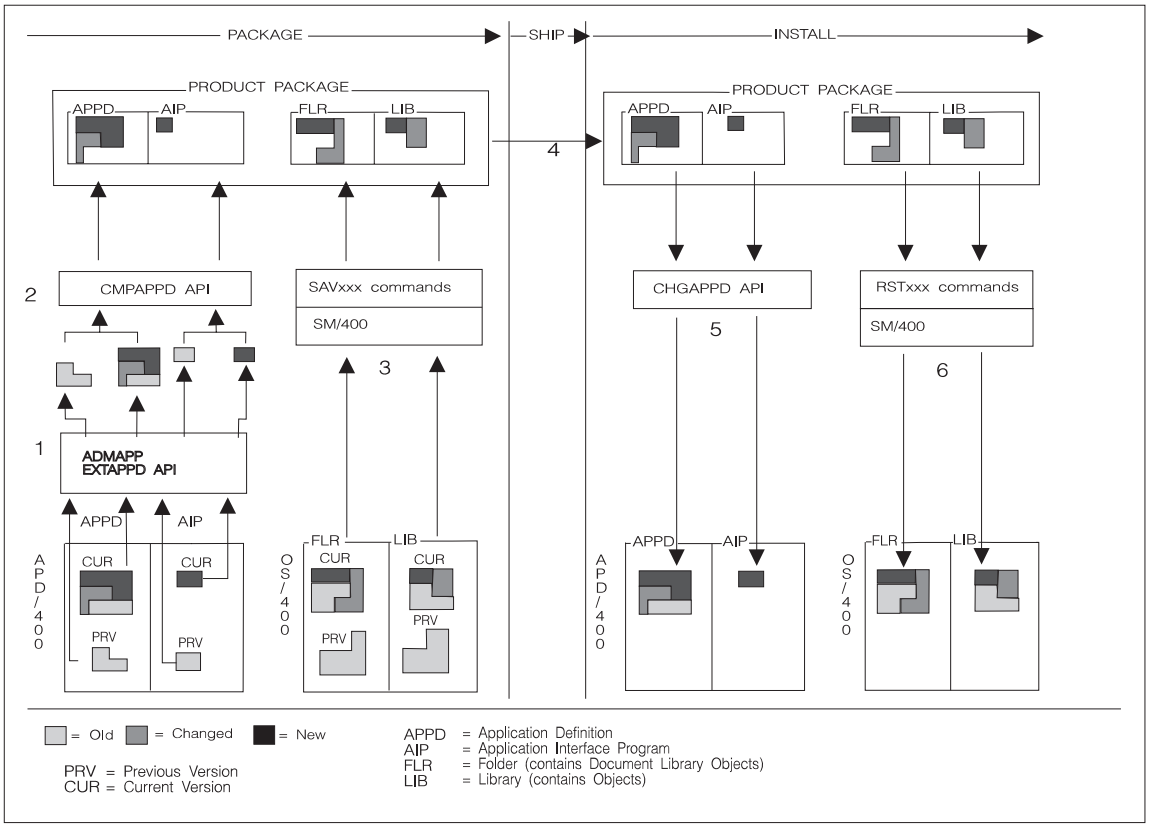

Figure 4. Package, Ship, and Install Application Updates

1 The application definitions and the AIP of both the previous and the current version of the application are extracted from the APD/400 data repository using one of the following methods:

1. Option `3` (Copy to library QAPDIAHDR) of the `Administer Applications (Developer)` function (ADMAPP). This function uses the library name QAPDIAHDR for the library that receives the application definitions.
2. The Extract Application Definitions API (EXTAPPD).

2 The previous and the current versions are compared using the Compare Application Definitions API (CMPAPPD). The result is a description of the changes (that is, the previous and the current version for every changed item) between the two versions that can be included in the product package.

3 As stated earlier, APD/400 has no complete concept for packaging or installing updates to the OS/400 part of an application. This is a task for software developers who can use their own software or

SystemManager/400. A discussion of this task goes beyond the scope of this book.

4 The update package can be shipped on any physical media (tape, cartridge, diskette, or CD-ROM), or electronically as a savefile.

In contrast to the standard product package, an update product package need not follow APD/400 conventions, that is, the way objects are saved on media is determined by the requirements of your individual procedures.

5 Using the Change Application Description API (CHGAPPD), the APD/400 part of the application is updated. This API reads the description of changes from the update product package, reads the application definitions of the previous version, and updates the previous version (changes modified items and adds new items), while preserving (as far as possible) the changes that have been made on the target system since the application was installed.

The CHGAPPD API can also be used to create a report of potential changes to the previous version on the target system before the update is processed. Based on this report, a decision can be made as to whether or not the update should be installed. The user can also change the application definitions (that is, rename items) before updating the application.

6 The OS/400 part of the application is updated. For example:

- New and changed programs are copied to the production libraries
- Database files are changed and existing data is migrated
- DLOs are modified.

Creating the Standard Product Package

If your application is to be installed on the target system using the `Install Applications` function (the standard method), you must do the following:

- Use the `Administer Applications (Developer)` function to create a list of library descriptions used in your application.
- Create a library named QAPDIAHDR that contains the application definitions of your application.
- Create an installation tape that contains the QAPDIAHDR library, the online Help folder (if applicable), and the application libraries.

These tasks are described in the following.

Creating a List of Application Library Descriptions

To create a list of application libraries, start the `Administer Applications (Developer)` function by entering the expert code ADMAPP. On the `Administer Applications (Developer)` display, select option `12` (Work with Application Libraries) for your application. The following is displayed:

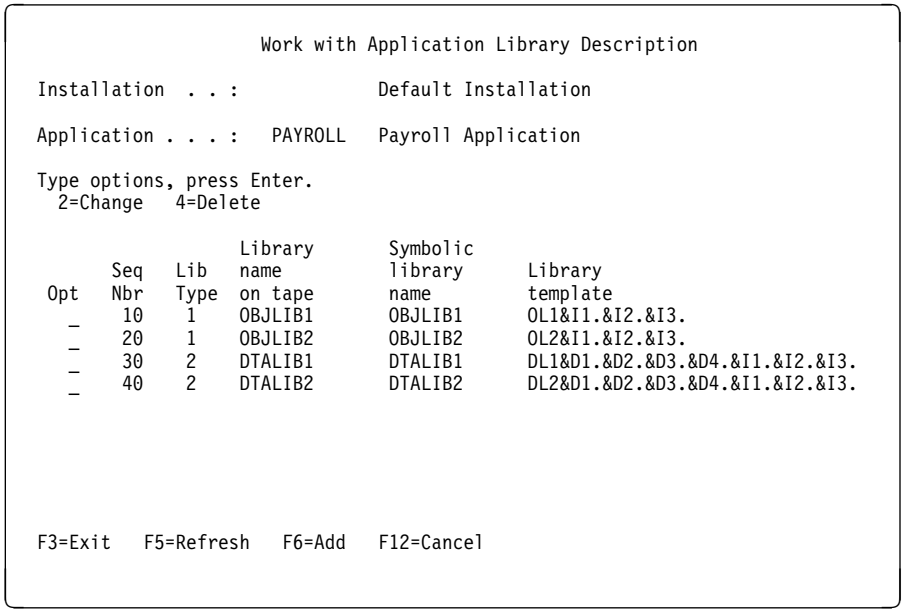

Figure 5. Work with Application Library Description

Limitations of Administer Applications (Developer)

The `Administer Applications (Developer)` function has been designed to create an installation package. You can set up the list of libraries only for an application that is to be installed on a different system. However, you **cannot** use this function to modify a list of libraries for an application that has been installed using the `Install Applications` function.

Adding an Application Library Description

To add a description of a library that is to be restored from tape when your application is installed, press F6. The following is displayed:

```
                    Add Application Library Description

 Installation  . :             Default Installation

 Application . . :   PAYROLL   Payroll Application

 Type choices, press Enter.

   Library sequence number . . . .    30

   Library type  . . . . . . . . .    2        1=Object
                                               2=Data
                                               3=Source
   Library name on the
     installation tape . . . . . .    DTALIB1___

   Symbolic library name . . . . .    DTALIB1___

   Library name template . . . . .    DL1&D1.&D2.&D3.&D4.&I1.&I2.&I3.


 F3=Exit   F12=Cancel
```

Figure 6. Add Application Library Description

Enter the following information:

`Library sequence number`
: The relative position of this library on the tape. The libraries are restored from the tape using the sequence numbers specified in this field. You can leave gaps in your numbering sequence.

`Library type`
: The library type. Specify `1` if the library contains objects, `2` if it contains data, or `3` if it contains files. If you want to support multiple data sets in your application, store data and objects in different libraries and indicate that here.

`Library name on the installation tape`
: The name of the library on the installation tape.

To understand the following descriptions of symbolic library name and library name template, you must have some knowledge of the multiple concept of APD/400. “Concept of Installations, Applications, and Data Sets” on page 1 describes the installation, application, and data-set concepts of APD/400, how to design an application to be multiinstallable and multi-data set enabled, and how symbolic library names and library name templates can be used for this support.

`Symbolic library name`
: The symbolic library name without the leading ampersand (&).

  Note: The name must be unique within the list of libraries for your application, and must follow OS/400 naming conventions. Maximum length for the symbolic library name is nine characters.

  You can use a symbolic library for:

  - The library that contains Help displays or panel groups (see “Describing the Application” on page 9).
  - The library for an object entry of an object exclusion list. Refer to the *IBM Application Program Driver/400 Version 3: Administrator's Guide* for more information on exclusions.

Library name template
: The name template for this library. If you do not provide support for multiple data sets or for multiple installations, specify the name of the library in the template.

  If you support multiple data sets or multiple installations in your application, specify the library naming rules to be used by the Install Applications functions at application installation time.

  The name template can contain:

  1. A-Z, $, or # (Hex 7B) for the first byte.
  2. A-Z, 0-9, $, # (Hex 7B), _ or . for bytes 2 through 10.
  3. A placeholder for a byte of the installation ID (&I1., &I2., and &I3.).
  4. A placeholder for a byte of the data-set ID (&D1., &D2., &D3., and &D4.).

When an authorized user selects Install Applications, the user is asked to specify installation and data-set IDs. If a library is restored by APD/400 from your installation tape, the name of that library in APD/400 is created from the naming rules set up in the Library name template and the specified values for installation ID and data-set ID.

Notes:

1. To support multiple data sets, data libraries are created for each data-set ID. Only libraries with the library type 2 (Data) can contain placeholders for data sets.
2. Blanks can be specified for installation ID or data-set ID when your application is installed. If you specify placeholders in the middle of the library name template, library names that are not valid in OS/400 could be created.
3. If your application supports multiple data sets or installations, you must create the correct library list in the AIP of your application. Make sure that when you specify or change a library name template in this field, the AIP supports the template.

   Refer to “Creating an AIP” on page 13 for a description of how to create an AIP.
4. If the library defined here is the library that contains the Help displays, this library must not contain any &Dn. placeholders.

Changing an Application Library Description

To change the description of an application library, specify 2 (Change) in the corresponding option column on Work with Application Library Description. Revise Application Library Description is displayed.

Deleting an Application Library Description

To delete an application library description, specify 4 (Delete) in the corresponding option column on Work with Application Library Description. On Delete Application Library Description, press Enter to confirm the deletion.

Creating the QAPDIAHDR Library

There are two methods you can use to extract the definitions of your application to the QAPDIAHDR library:

1. The Extract Application Definitions API (EXTAPPD). Specify QAPDIAHDR as the target library to receive the extracted application definitions or use another library name and rename it afterwards to QAPDIAHDR using the RNMOBJ command.

   "EXTAPPD Extract Application Definitions" on page 84 describes this API, and "Sample Scenarios" on page 45 contains samples that use this API.

2. The `Administer Applications (Developer)` function. This is an interactive function that guides you through the creation of the QAPDIAHDR library. To select this function, enter ADMAPP, and on the `Administer Applications (Developer)` display select option `3` (Copy to library QAPDIAHDR) for your application.

   When you select this function, APD/400 creates a library with the name QAPDIAHDR that contains all the application definitions that belong to your application.

   If QAPDIAHDR already exists, a confirmation window is displayed before APD/400 overwrites data in that library. You can select to overwrite the data, to retry the request (for example, after you have renamed the existing library), or to cancel the process.

Creating the Installation Tape

When you have created the QAPDIAHDR library containing the APD/400 application definitions, you can create the installation tape. Because this tape is intended to be used by the `Install Applications` function, it must have the following structure:

Note: This structure describes general-use programming interface and associated guidance information.

Table 4. Structure of the Installation Tape

Sequence Number	Library/Folder	Description
1	QAPDIAHDR	A library containing the APD/400 application definitions (stored in database files named QAAFxxxxA) and the AIP. This is also the library in which the Post-Installation user exit should be stored.
2	Folder	This is the folder that contains the online Help documents when you specify Help type 1 (OfficeVision/400 documents) for your menus. Otherwise it can be omitted.
3	Application libraries	The application libraries must be saved in exactly the same sequence as specified in the application library description.

Except for QAPDIAHDR (which must have a sequence number of 1), the sequence numbers do not have to be the same as those in Table 4. However, the relative sequence must always be:

```
QAPDIAHDR ──► Folder ──► Application library
```

This is because the `Install Applications` function searches the tape in the order: folder (if specified), application libraries.

The Post-Installation user exit can be stored in any of the application libraries defined for your application, but it is recommended to store it in QAPDIAHDR, as this is the first library searched.

The following describes the procedure to create the installation tape:

1. Initialize the tape using the command INZTAP. No special volume or owner IDs are required. The APD/400 `Install Applications` function supports every tape media that is supported by the AS/400.
2. Save library QAPDIAHDR to the tape, specifying not to rewind the tape. For example:

   ```
   SAVLIB LIB(QAPDIAHDR) DEV(TAP01) ENDOPT(*LEAVE)
   ```

   The library QAPDIAHDR must be the first library on the tape because `Install Applications` tries to restore the library from the first file (sequence number 1) on the tape.
3. If you have specified a folder to store the Help texts of your menus, save the folder to the tape (again, specifying not to rewind the tape). For example:

   ```
   SAVDLO DLO(*ALL) FLR(YOURFLR) DEV(TAP01) ENDOPT(*LEAVE)
   ```
4. Save the application libraries in the sequence you specified with the `Library sequence number` field. For example:

   ```
   SAVLIB     LIB(YOURLIB1) DEV(TAP01) ENDOPT(*LEAVE)
   ```

   If you have more than one library, you can write a short program to store the libraries in the correct sequence. The following is an example of such a program:

   Note: This interface is as described in “Interfaces for Version 3 Release 6” on page viii.

   ```
   /*-------------------------------------------------------------------*/
   /*   Program to save the application libraries to the                */
   /*   installation tape.  The list of libraries to be                 */
   /*   saved is in QAPDIAHDR/QAAFLIBRA.                                */
   /*-------------------------------------------------------------------*/
   PGM
   DCLF FILE(QAPDIAHDR/QAAFLIBRA)

   /*-------------------------------------------------------------------*/
   /* Save the libraries                                                */
   /*-------------------------------------------------------------------*/
   SAVLIB:
     RCVF
     MONMSG     MSGID(CPF0864) EXEC(GOTO SAVLIBEND)
     SAVLIB     LIB(&LIBSAV) DEV(TAP01) ENDOPT(*LEAVE)
     GOTO       SAVLIB
   SAVLIBEND:

   ENDPGM
   ```
5. Unload the tape or save the next application to the tape. For example:

   ```
   CHKTAP     DEV(TAP01) ENDOPT(*UNLOAD)
   ```

6. Delete library QAPDIAHDR:

```
DLTLIB     LIB(QAPDIAHDR)
```

Notes:

1. You could also write a CL program to process these commands. This would be useful if you frequently create installation tapes.
2. If you want to store more than one APD/400 application on the same tape, all applications must have QAPDIAHDR as the first library. In this case, only the first QAPDIAHDR library on the tape must have a sequence number of 1. Use the OS/400 RNMOBJ command to store different QAPDIAHDR libraries on the same tape.

Sample Scenarios

As stated earlier, the APD/400 standard method is not the only method that you can use to install your application. You can also use the APIs supplied with APD/400. These APIs must be used for the installation as well as for the updating of an existing application. Using the APIs to extract, install, change, compare, and delete APD/400 application definitions, you can build your own individual procedures for base installation and update of your application.

The APIs are described as follows:

"CHGAPPD Change Application Definitions" on page 69
"CMPAPPD Compare Application Definitions" on page 79
"DLTAPPD Delete Application Definitions" on page 80
"DSPINSAPP Display Installed Applications" on page 81
"EXTAPPD Extract Application Definitions" on page 84
"INSAPPD Install Application Definitions" on page 85
"WRKDST Work with Data Sets" on page 93
"WRKINS Work with Installations" on page 95.

Two models (or scenarios) describe how the APD/400 functions and APIs can be used to package, ship, and install software designed to run under control of APD/400.

Note: The models do not describe the installation of the OS/400 part (for example, install libraries and folders, and create user profiles) of the application (application programs or database files). Use SystemManager/400 or OS/400 commands to install this part.

The source code for the programs used in the models is shipped with APD/400 in the source files QAPD/QAAFSMPL0 and QUSRSYS/QAAFSMPL0.

Model 1: Centrally Maintained Software

This model describes a solution for a company that has developed an application to be used in its branch offices. Because employees in the branch offices do not have the necessary data processing skill, the software is maintained from a central site (the head office). Figure 7 on page 46 shows the model:

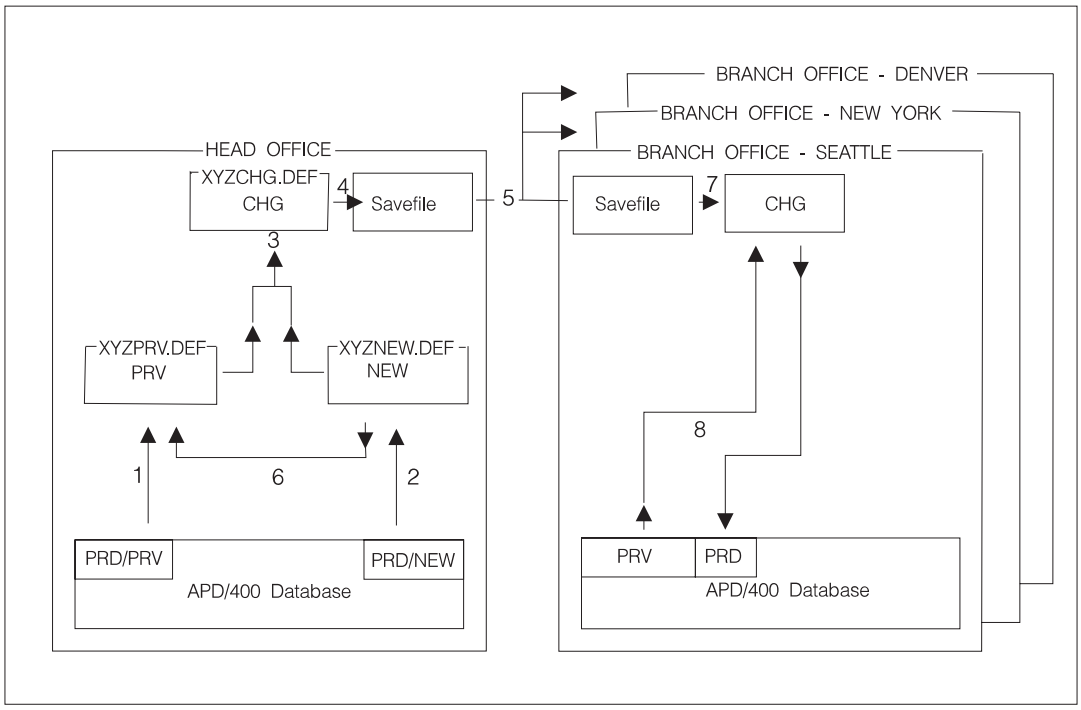

Figure 7. Model 1: Application Update

This model consists of the programs APIM1P1, APIM1P2, and APIM1P3:

APIM1P1:

1 The application definitions of the previous production version of the application are extracted into the file XYZPRV.DEF using the EXTAPPD API. This version is referred to as the PRV version.

APIM1P2:

2 The application definitions of the new production version of the application are extracted into the file XYZNEW.DEF using the EXTAPPD API. This version is referred to as the NEW version.

3 The CMPAPPD API is used to compare the PRV and the NEW versions. The differences are stored in the XYZCHG.DEF library.

4 The XYZCHG.DEF library is saved to a savefile.

5 The savefile is sent to the branch offices.

6 The library XYZNEW.DEF is made the previous version in library XYZPRV.DEF.

APIM1P3:

7 The savefiles sent from the head office are received and the library XYZCHG.DEF is restored.

8 The CHGAPPD API is used to install the changes stored in the XYZCHG.DEF library to the PRV version. The application definitions in the branch offices are now the same as the NEW production version in the head office.

Notes:

1. With some simple variations you could use the same programs to install the application on the same system. For example, to transfer changes from a test to a production installation.
2. You could also use the EXTAPPD and INSAPPD APIs to copy an application from one installation to another.

Model 2: One Tape for Base Install and Update

In this scenario the install process is designed so that one tape can be used for both base installation and update of the software. To do this, a copy of the original version is stored on the customer's (target) system. The compare process is done on the target system (compare the new base version against the old original version).

This scenario also shows how an installation can be created automatically using APIs, and how other information can be retrieved from the APD/400 data repository. Figure 8 shows the model:

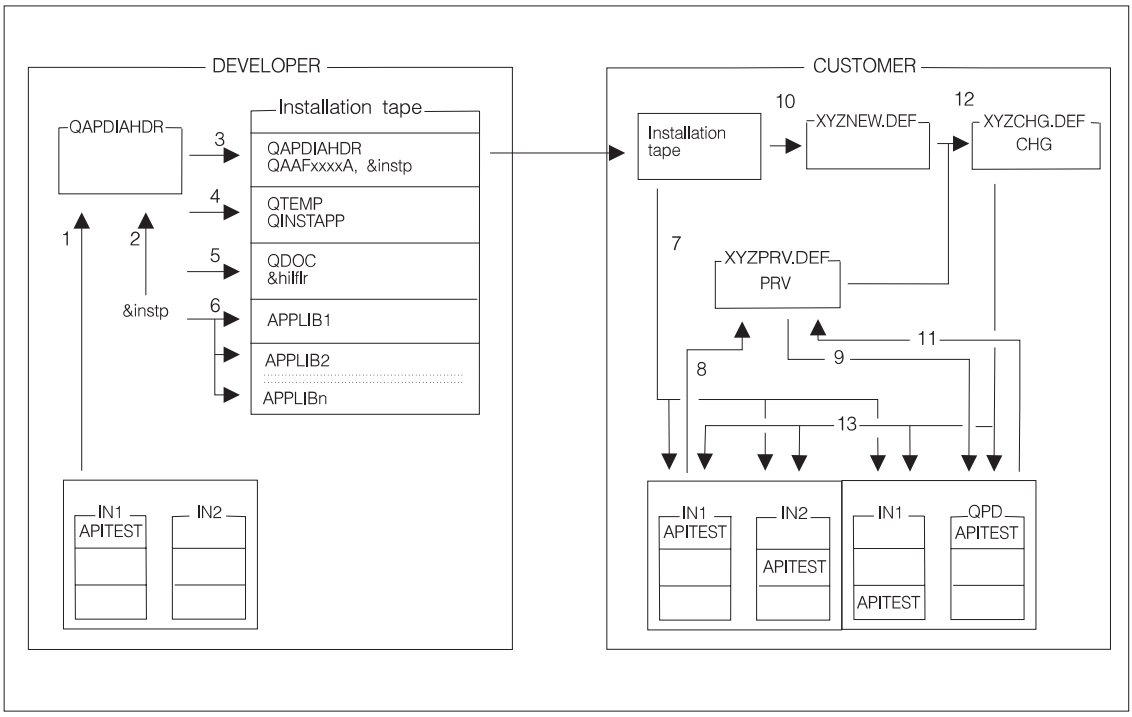

Figure 8. Model 2: Base Install and Update

The model consists of the programs APIM2P1, APIM2P2, APIM2P2A, and APIM2P3:

APIM2P2:
This program is used on the developer's site to create the installation tape.

1 The application definitions for application APITEST in installation `IN1` are extracted to library QAPDIAHDR.

2 The file QAPDIAHDR/QAAFANWG0 is read to determine the name of the Post-Installation user exit (`&instp`) of application APITEST. This program is also copied to library QAPDIAHDR.

Note: This interface is as described in "Interfaces for Version 3 Release 6" on page viii.

3 Library QAPDIAHDR is saved as the first file on the installation tape.

4 The second file is saved to the tape; it contains the LODRUN program QTEMP/QINSTAPP. This is a renamed copy of sample program APIM2P3.

5 The online Help folder is saved to tape. This is the third file saved, and it must be saved after the QAPDIAHDR library and before the application libraries.

APIM2P2A:

This is a subprogram of APIM2P2.

6 The list of libraries for application APITEST is read from file QAPDIAHDR/QAAFLIBRA. Every library found is saved to the installation tape.

Note: This interface is as described in "Interfaces for Version 3 Release 6" on page viii.

The `Install Applications` function is not part of the sample programs but is documented for completeness. In the sample, it is used for the base installation of application APITEST into the installations `IN1`, `IN2`, and `IN3`.

7 The installation package is read from the tape, new installations (`IN1`, `IN2`, and `IN3`) are created, and all application data is copied to the APD/400 data repository.

APIM2P1:

This program is used as the Post-Installation user exit for the `Install Applications` function (see "POSTINS Post-Installation" on page 60 for a description of this user exit). It receives the installation ID, application ID, and (if applicable) the data-set ID of the application currently installed from the `Install Applications` function.

8 The application definitions of the installed application are extracted to library XYZPRV.DEF (using the EXTAPPD API).

9 Information about data sets is retrieved (using the WRKDST API), a new installation called `QPD` is created (using the WRKINS API), and the application definitions are installed from library XYZPRV.DEF to the new installation (using the INSAPPD API).

APIM2P3:

This program is used for updating application APITEST. It is stored on the installation tape (created by program APIM2P2) with the name QTEMP/QINSTAPP, and can be loaded and started using the OS/400 command LODRUN.

10 The library with the new definitions is restored from tape to library XYZNEW.DEF.

11 The application definitions of the previous version are extracted from installation `QPD` to library XYZPRV.DEF using the EXTAPPD API.

12 The previous and the new application definitions are compared (using the CMPAPPD API), and the results are stored in library XYZCHG.DEF.

13 A list of all installations where application APITEST is currently installed is created (using the DSPINSAPP API), and the application definitions for all occurrences of application APITEST are updated (using the CHGAPPD API). This is done for the installations `IN1`, `IN2`, `IN3`, and `QPD`.

Considerations for Multilingual Support

As well as changes necessary to the AIP (see "Allowing for Multilingual Support" on page 24), consider the following points when packaging, shipping, and installing an application enabled for multilingual support:

- APD/400 does not currently support the installation of textual data objects (Help documents, panel groups, display files, and message files). You must provide the programs to create the necessary QAPDxxxx libraries and QAFxxxx folders, and copy the textual data objects from tape to the library or folder of each language to be installed.
- When `*MRI` is specified for the `Folder/Library/User exit` field of the application description (see the fifth page of `Add Application Entry` on Page 11), APD/400 does not automatically install this library or folder (as it does for Help folders of applications not enabled for multilingual support).
- When textual data stored in message files is modified (for example, during a release change), you must use APD/400 to rebuild all menus that use these messages (using the &msg symbol). To do this, run the APD/400 reorganization with a reorganization level of 4 or greater.

For more information on the APD/400 multilingual support features, see the appendix on "Multilingual Support" in *IBM Application Program Driver/400 Version 3: Administrator's Guide*.

Chapter 4. User Exits and APIs

This chapter describes:

- APD/400 user exits and APIs, including a listing of the parameters used in the interface for each user exit or API
- General-use programming interface and associated guidance information.

The following diagrams show sample call structures during the processing of APD/400 to explain the terms “user exit” and “API” as they are used in this chapter.

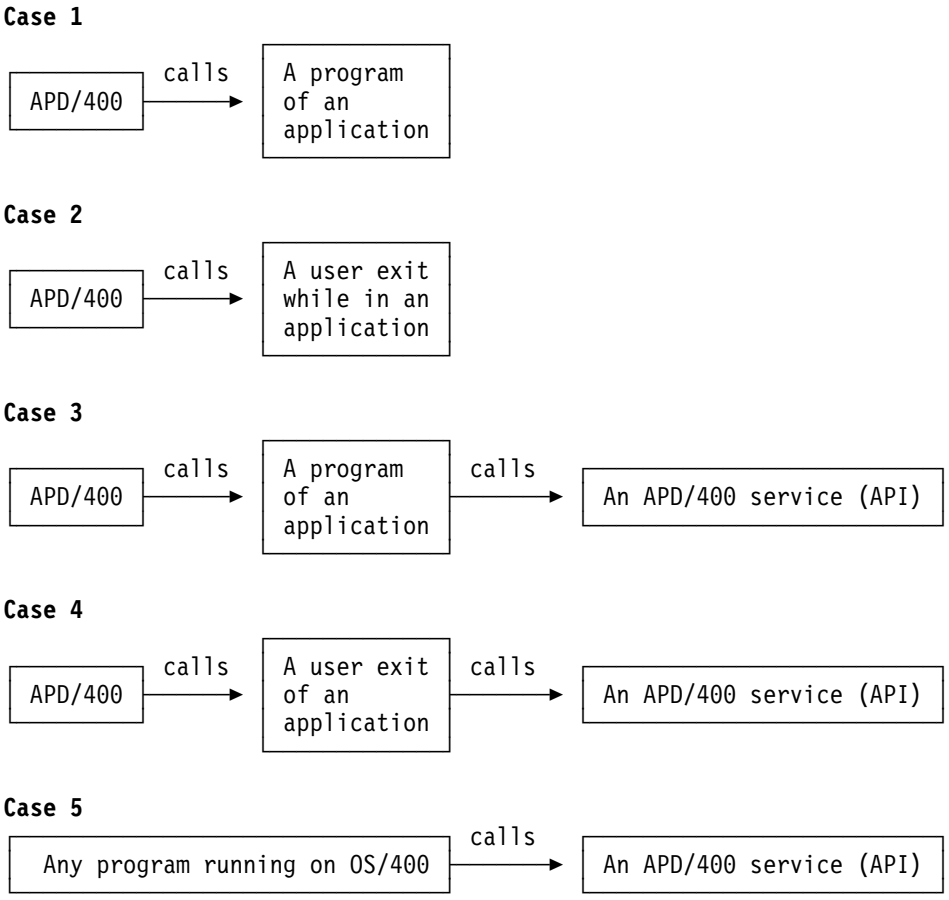

Figure 9. APD/400 Call Structures

Application programs are called in various situations by APD/400 (calls on the left-hand side in Figure 9). In most cases, an application program is called when a user has selected a task from an APD/400 menu (cases 1 and 3).

Application programs can also be called by APD/400 to allow the application to enhance services normally provided by APD/400 functions, or to do additional checking or provide services not part of APD/400 functions. In these situations, the application program is called a user exit (cases 2 and 4).

An application developer defines names of user-exit programs for the application being developed using APD/400 administrative functions. “User-Exit Descriptions” on page 53 describes the instances in APD/400 when user exits are called by

APD/400, if so defined by the application or user, and also provides information about the parameters passed from APD/400 to user exits.

Certain APD/400 functions can be called from application programs, whether processing under control of APD/400 (being called as a task from a menu) or not (calls on the right-hand side in Figure 9 on page 50). Each function has a unique name. A function name and the description of the parameters passed with it is called an API (cases 3, 4, and 5). A parameter description for each function is given in "API Descriptions" on page 66.

User Exits

The following describes how user exits are called, the user-exit communication areas passed from APD/400 to the AIP, the messages produced by user exits, and lists the user exits supplied with APD/400.

Calling User Exits from APD/400

A user-exit program is part of an application, and is called via the AIP as is any application program. An exception to this is the Post-Installation user exit (POSTINS), which is called directly from APD/400, and not via the AIP.

The AIP parameter contains two fields that are used for the communication between APD/400 and the user-exit program:

PTYPE If this field is set to `X`, a user-exit program must be called by the AIP.

FRFLD This structure serves as the communication area for the call of a user exit, and contains all information necessary to control its processing.

The user-exit communication area is initialized by APD/400 before each call to a user-exit. It consists of two parts:

1. The user-exit-independent part contains information valid for all user exits, for example, the name of the user-exit program. This part is described in "User-Exit Communication Area."
2. The user-exit-dependent part contains information required only by the individual user exit. This part is described individually for each user exit.

Note: User exits introduced with Version 1 Release 2 of APD follow the described interface structure. User exits available with the previous release of APD (Version 1 Release 1) still use the original interface. Which interface layout is applicable is explained under the description of each user exit.

User-Exit Communication Area

The user-exit communication area is passed by APD/400 to the AIP in the field AIP.FRFLD. The structure of the user-exit-independent part of the communication area is as follows:

Table 5. User-Exit-Independent Part of the Communication Area

Parameter	Use	Type	Len	Pos	Description
1 User-exit name UEXNAM	INPUT	CHAR(10)	10	1	User-exit ID. Filled by APD/400 before calling an application program. This ID can be used in the AIP to process actions specific for each user exit. It can also be used in a user-exit program to double check that the program was called from the expected function in APD/400. This field is blank when an application program is called that is not a user exit.
2 Completion code UEXCCD	OUTPUT	CHAR(2)	2	11	Completion code. Initialized to 00 by APD/400. Allows the user-exit program to communicate with APD/400. Should always be set by the user-exit program. The way the completion code is interpreted by APD/400 is described under each user exit.
3 Reserved			4	13	This space is reserved for future use.
4 User-exit program name UEXPGM	INPUT	CHAR(10)	10	17	This is the name of the program to be called by the AIP. As this program is not directly called by APD/400, it could also be interpreted by the AIP as the name of a command or the name of a REXX procedure.
5 Reserved			2	27	This space is reserved for future use.
6 User-dependent part	INPUT	CHAR(172)	172	29	This part of the user-exit communication area is reserved for the specific information used by each individual user exit. It is described individually for each user exit.
Total			**200**		

Note: If a user exit is called from APD/400, feedback is via the field UEXCCD. If an application program is called from APD/400, feedback is via the field ERROR, which allows a possible restart condition to be specified.

Messages from User Exits

In addition to the completion code UEXCCD, a user exit may need to return text to the calling APD/400 function, to be displayed by APD/400. This is done in the same way as with messages sent from any OS/400 program; all messages are sent back to the message queue of the APD/400 program that called the AIP. To do so, you must code a loop in your AIP as in the following:

```
/* DO UNTIL < no more messages in the program msg.queue >  */
LOOP99BEG:
  RCVMSG     MSGDTA(&MSGDTA) MSGID(&MSGID) MSGF(&MSGF) +
             MSGFLIB(&MSGFLIB)
  IF (&MSGID *EQ ' ') +
  THEN(DO)
    GOTO LOOP99END
  ENDDO

  SNDPGMMSG  MSGID(&MSGID) MSGF(&MSGFLIB/&MSGF) +
             MSGDTA(&MSGDTA) TOPGMQ(*PRV)

  GOTO LOOP99BEG
LOOP99END:
```

For example, if a user exit is called to determine the application-defined exclusion status of a program that the user wants to start, the user exit may return a return code indicating that the program cannot be started. In this case, APD/400 does not start the program the user has requested. The reason that the program cannot be started is determined only by the user exit. The user exit can generate the text that can then be displayed by APD/400 using the method described.

User-Exit Descriptions

The following are the user exits supplied with APD/400.

ADMDSTE Administer Data-Set Entries

This user exit enables the application developer to perform certain application-dependent activities if a data set is created, changed, or deleted using the function `Administer Data Sets`.

You can define this user exit for your application using the `Administer Applications (Developer)` function (ADMAPP).

Interface Description

Table 6 (Page 1 of 2). Interface for the Administer Data-Set Entries User Exit (ADMDSTE)

Parameter	Use	Type	Len	Pos	Description
1 User-exit name UEXNAM	INPUT	CHAR(10)	10	1	ADMDSTE The name of the user exit for Administer Data-Set Entries.
2 Completion code UEXCCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **Other** An error occurred during the processing of the AIP or the user-exit program. The results are unpredictable. `Administer Data Sets` is redisplayed with the option field of the item selected in reverse image. All messages sent through the AIP are displayed in the message line. The data-set entry is not updated.
3 Reserved			4	13	This space is reserved for future use.

Table 6 (Page 2 of 2). Interface for the Administer Data-Set Entries User Exit (ADMDSTE)

Parameter	Use	Type	Len	Pos	Description
4 User-exit program name UEXPGM	INPUT	CHAR(10)	10	17	The name of the program to be called by the AIP.
5 Reserved			2	27	This space is reserved for future use.
6 Installation name INSID	INPUT	CHAR(3)	3	29	The name of the installation to which the data set to be maintained belongs.
7 Application name ANWID	INPUT	CHAR(7)	7	32	The name of the application to which the data set to be maintained belongs.
8 Data-set name FIRNR	INPUT	CHAR(4)	4	39	The name of the data set to be maintained.
9 Data-set description FIRNM	INPUT	CHAR(40)	40	43	The description of the data set to be maintained.
10 Operation code OPRCD	INPUT	CHAR(1)	1	83	Defines the type of administration requested when this user exit is called: **1** Add data set. **2** Change data set. **3** Delete data set.
11 Return code RETCDE	OUTPUT	CHAR(2)	2	84	**00** Data-set entry changed. The data-set entry is updated and processing continues (`Administer Data Sets` is displayed or the next entry is processed). **03** F3 (Exit) pressed. The `Administer Data Sets` program has ended. The data-set entry is not updated. **12** F12 (Cancel) pressed. The data-set entry is not updated and processing is interrupted; `Administer Data Sets` is displayed and the cursor is on the corresponding entry in the subfile. For other return codes, no update is performed but processing continues.
Total			**85**		

BCHPRM Overwrite Batch Task Parameter

This user exit enables a parameter program to be called when a user invokes a batch job, and allows parameter data to be entered for the batch job that is different from that defined in APD/400 for the batch task.

The user-exit information can be maintained using the `Administer Menus` (ADMMNU) function.

Interface Description

The interface for this user exit is the input parameter of the AIP. In the interface description, only those fields are mentioned that are necessary for the user exit. All other fields of the AIP interface are not used and not mentioned in this description.

Note: The layout of this user-exit interface is the same as in Version 1 Release 1 of APD. No user-exit communication area is used here.

The value in the Pos (Position) column describes the relative position of the field in the AIP interface.

Table 7. Interface for the Batch Parameter User Exit (BCHPRM)

Parameter	Use	Type	Len	Pos	Description
1 Installation name AIP.INSID	INPUT	CHAR(3)	3	1	The name of the installation to which the task to be scheduled belongs.
2 Task name AIP.MTASK	INPUT	CHAR(10)	10	187	The name of the task to be scheduled.
3 Program name AIP.MPGMN	INPUT	CHAR(10)	10	243	The name of the user-exit program to be called by the AIP.
4 Application name AIP.ANWID	INPUT	CHAR(7)	7	303	The name of the application to which the task to be scheduled belongs.
5 Program type AIP.PTYPE	INPUT	CHAR(1)	1	311	This is set to the constant P by APD/400 to indicate that this is the BCHPRM user exit for Version 1 Release 1.
6 Program parameter AIP.MPPRT	UPDATE	CHAR(512)	512	312	The parameter that was defined with the Administer Menus function is passed to the user-exit program, which overwrites this parameter with its own values. The new value is then passed to the task when it is processed by APD/400's batch handler.
7 Schedule time AIP.DATTIM AIP.FRFLD(1-14)	UPDATE	CHAR(14)	14	864	Date and time when the job is to be processed, in the format YYYYMMTTHHMMSS (bytes 1 to 14 of field FRFLD).
8 Return code AIP.RETCDE AIP.FRFLD(32-33)	OUTPUT	CHAR(2)	2	895	Return code used to inform APD/400 how the user-exit program has ended. The following values are valid: **00** Successful completion: the task is scheduled with the specified date and time, and task parameter as passed from the user exit. **03** F3 (Exit) has been pressed in the user-exit program: the task scheduling is canceled. **12** F12 (Cancel) has been pressed in the user-exit program: the task scheduling is canceled. These are bytes 32 to 33 of field FRFLD.
Total			**559**		

CHKAUT Check Authorization

This user exit enables the application developer or user to have application-dependent authorization checking without having to use the APD/400 authorization concept. The user-exit program determines the authorization level described by:

- Installation name
- Application name
- Authorization list name
- User name.

Authorization checking in APD/400 is accomplished using authorization lists. There are two types of authorization list:

- Normal lists point to a list of users and user groups with corresponding authorization levels.
- User exits point to an application program that is processed whenever the authorization level for the authorization list is requested. You can define one user-exit program for each authorization list. This procedure is not called if use authorization checking=N.

You can define both types of authorization list with the function `Administer Authorization Lists` (ADMAUT).

Interface Description

Table 8 (Page 1 of 2). Interface for the Check Authorization User Exit (CHKAUT)

Parameter	Use	Type	Len	Pos	Description
1 User-exit name UEXNAM	INPUT	CHAR(10)	10	1	CHKAUT The name of the Check Authorization user exit.
2 Completion code UEXCCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **Other** An error has occurred during the processing of the AIP or the user-exit program. The results are unpredictable. All messages sent by the AIP are displayed on the next display shown by APD/400. Authorization level (BRSTF) 0 is assumed.
3 Reserved			4	13	This space is reserved for future use.
4 User-exit program name UEXPGM	INPUT	CHAR(10)	10	17	The name of the program to be called by the AIP.
5 Reserved			2	27	This space is reserved for future use.
6 Installation name INSID	INPUT	CHAR(3)	3	29	The name of the installation to which the authorization list to be checked belongs.
7 Application name ANWID	INPUT	CHAR(7)	7	32	The name of the application to which the authorization list to be checked belongs.

Table 8 (Page 2 of 2). Interface for the Check Authorization User Exit (CHKAUT)

Parameter	Use	Type	Len	Pos	Description
8 Authorization list name BRFKT	INPUT	CHAR(10)	10	39	The name of the authorization list that points to the user exit to be processed.
9 User name USRID	INPUT	CHAR(10)	10	49	The user profile of the user that owns the current job as retrieved from OS/400 using RTVJOBA.
10 Authorization level BRSTF	OUTPUT	CHAR(1)	1	59	The authorization level as a 1-character field (0 through 9). This field must be set by your user-exit program. APD/400 interprets: **0** User not authorized. **1-9** Authorization level of the user.
Total			**59**		

CHKEXC Check Exclusion

This user exit enables the application developer or user to have exclusion checking without using the APD/400 exclusion concept.

The user-exit program for exclusion checking can be defined with the function Administer Exclusions (ADMEXC). It is possible to have an unlimited number of exclusion definitions and therefore an unlimited number of different user-exit programs for every exclusion list.

Control can be passed to this user exit when a task is to be processed. APD/400 checks if this task belongs to an exclusion list (type *FCT), and if there is any exclusion record defined for this exclusion list. If one exists with exclusion type 3 (user exit), APD/400 calls the AIP of the corresponding application with the appropriate interface. This procedure is not called if use exclusion checking=N.

Interface Description

The interface for this user exit is the input parameter of the AIP. In the interface description, only those fields are mentioned that are necessary for the user exit. The value in the Pos (Position) column describes the relative position of the field in the AIP interface. All other fields of the AIP interface are not used and therefore not mentioned in this description.

Table 9 (Page 1 of 2). Interface for the Check Exclusion User Exit (CHKEXC)

Parameter	Use	Type	Len	Pos	Description
1 Installation name AIP.INSID	INPUT	CHAR(3)	3	1	The name of the installation to which the task to be processed belongs.
2 Data-set name AIP.FIRNR	INPUT	CHAR(4)	4	91	The name of the currently active data set for the application to which the task to be processed belongs.
3 Task name AIP.MTASK	INPUT	CHAR(10)	10	187	The name of the task to be processed.

Table 9 (Page 2 of 2). Interface for the Check Exclusion User Exit (CHKEXC)

Parameter	Use	Type	Len	Pos	Description
4 User-exit program name AIP.MPGMN	INPUT	CHAR(10)	10	243	The name of the user-exit program to be called by the AIP.
5 Exclusion list AIP.PGMGR AIP.MPPRA(1..10)	INPUT	CHAR(40)	10	253	This field contains the name of the exclusion list to which the task to be processed belongs. Only the first 10 bytes of this field are used.
6 Error flag AIP.ERROR	OUTPUT	CHAR(1)	1	301	In the error flag, the user exit returns the information on whether the task can be processed or not: **0** No exclusion; the task can be processed. **<>0** The task is currently excluded.
7 Application name AIP.ANWID	INPUT	CHAR(7)	7	303	The name of the application to which the task to be processed belongs.
8 Program type AIP.PTYPE	INPUT	CHAR(1)	1	311	The constant X is passed in this field to the user exit.
Total			**46**		

DSPHLP Display Help

This user exit enables an application to use its own programs and commands to display a task-oriented Help to the user. You can define one user exit for each application installed under APD/400 using the Administer Applications (Developer) function (ADMAPP).

The Help flag must be set to 4=User exit and the name of the user-exit program is stored in the Library/Folder/User Exit field. Make sure that only the first ten bytes are used. This name is passed in the parameter field UEXPGM to the AIP.

Interface Description

Table 10 (Page 1 of 3). Interface for the Display Help User Exit (DSPHLP)

Parameter	Use	Type	Len	Pos	Description
1 User-exit name UEXNAM	INPUT	CHAR(10)	10	1	DSPHLP The name of the Display Help user exit.
2 Completion code UEXCCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **Other** An error has occurred during the processing of the AIP or the user-exit program. All messages sent by the AIP are displayed on the next display shown by APD/400. No additional Help is shown.
3 Reserved			4	13	This space is reserved for future use.

Table 10 (Page 2 of 3). Interface for the Display Help User Exit (DSPHLP)

Parameter	Use	Type	Len	Pos	Description
4 User-exit program name UEXPGM	INPUT	CHAR(10)	10	17	The name of the program to be called by the AIP. Retrieved from the Library/Folder/User Exit field of the application definition.
5 Reserved			4	27	This space is reserved for future use.
6 Installation name INSID	INPUT	CHAR(3)	3	31	The name of the installation to which the task belongs for which the Help information has been requested.
7 Application name ANWID	INPUT	CHAR(7)	7	34	The name of the application to which the task belongs for which the Help information has been requested.
8 Task name MTASK	INPUT	CHAR(10)	10	41	The name of the task for which the Help information has been requested.
9 Identifier 1 HLDOC	INPUT	CHAR(10)	10	51	The first identifier that can contain information for the user-exit program to retrieve the task-oriented Help. This value is retrieved from the identifier 1 field of the task description, and can be changed using the Administer Menus function.
10 Identifier 2 HLLAB	INPUT	CHAR(10)	10	61	The second identifier that can contain information for the user-exit program to retrieve the task-oriented Help. This value is retrieved from the identifier 2 field of the task description, and can be changed using the Administer Menus function.
11 Language feature code LNGFC	INPUT	CHAR(4)	4	71	The language feature code of the currently used language for this application. The IBM language feature code in the form 29xx is used. Refer to the *AS/400 National Language Support Planning Guide* for a complete list of IBM language feature codes. This parameter is set only for applications that have multilingual support for online Help, that is, the Library/Folder/User Exit field on the application definition must be set to *MRI. For other applications, this parameter contains blanks.
12 Current column CURCOL	INPUT	ZONED(3)	3	75	The current column position of the cursor when the user presses the Help function key.
13 Current row CURROW	INPUT	ZONED(2)	2	78	The current row position of the cursor when the user presses the Help function key.
14 Upper row UPLROW	INPUT	ZONED(2)	2	80	The upper row of a “Do-Not-Cover-Area” that should not be used to display Help information.
15 Left column UPLCOL	INPUT	ZONED(3)	3	82	The left column of a “Do-Not-Cover-Area” that should not be used to display Help information.
15 Lower row LOWROW	INPUT	ZONED(2)	2	85	The lower row of a “Do-Not-Cover-Area” that should not be used to display Help information.

Table 10 (Page 3 of 3). Interface for the Display Help User Exit (DSPHLP)

Parameter	Use	Type	Len	Pos	Description
15 Right column LOWCOL	INPUT	ZONED(3)	3	87	The right column of a “Do-Not-Cover-Area” that should not be used to display Help information.
Total			**89**		

POSTINS Post-Installation

This user exit enables the application developer to perform certain application-dependent activities, such as to install libraries or folders, after installation of the application using the `Install Applications` function. You can define the Post-Installation user exit for your application using the `Administer Applications (Developer)` function (ADMAPP).

If the user-exit program is defined, it is called when the APD/400 part of the application installation (merge files, install libraries, and so on) is completed.

It is recommended to store this user-exit program in the QAPDIAHDR header library.

An error in this user exit can be signalled to APD/400 by sending an escape message to *PRV. Before sending the message, any task already completed by the user exit is reversed, for example, any libraries, journals, or user profiles that have been deleted are restored.

Interface Description

The interface for the Post-Installation user exit is different from other user-exit interfaces in terms of how the parameters are passed to the user exit program:

1. The user-exit program is called directly from the APD/400 application installation procedure and not via the AIP. Because an AIP is not used, the library list, local data area, and so on, are the responsibility of this program.
2. The parameters passed to the program are passed as single parameters, and not as one parameter represented by a structure.

Table 11 (Page 1 of 2). Interface for the Post-Installation User Exit (POSTINS)

Parameter	Use	Type	Len	Description
1 Installation name INSID	INPUT	CHAR(3)	3	The name of the installation in which the application is installed. This is defined by the user during application installation.
2 Application name ANWID	INPUT	CHAR(7)	7	The name of the application that is currently being installed.
3 Data-set name FIRNR	INPUT	CHAR(4)	4	The name of the data set that has been defined as the initial data set during application installation. If an application is multi-data set enabled, the user must define the name of the initial data set.
4 Device name DEV	INPUT	CHAR(10)	10	The device name specified during installation.

Table 11 (Page 2 of 2). Interface for the Post-Installation User Exit (POSTINS)

Parameter	Use	Type	Len	Description
5 Alias name ALIAS	INPUT	CHAR(7)	7	The alias name of your application specified during installation.
6 Type of install INSTP	INPUT	CHAR(1)	1	0 for initial installation of the application, or 1 for subsequent installation of a data set for an application already installed.
Total			**32**	

SAVRST Save/Restore

This user exit enables the developer of an application to decide whether a library about to be saved or restored can be processed or not, or to perform certain application-dependent activities before or after the save or restore. You can define one user exit for each application installed under APD/400 using the Administer Applications (Developer) function (ADMAPP).

A library that has been defined in the save and restore control records but that does not belong to any application installed under APD/400 (for example, a user library) cannot have an associated Save/Restore user exit.

Interface Description

Table 12 (Page 1 of 2). Interface for the Save/Restore User Exit (SAVRST)

Parameter	Use	Type	Len	Pos	Description
1 User-exit name UEXNAM	INPUT	CHAR(10)	10	1	SAVRST The name of the Save/Restore user exit.
2 Completion code UEXCCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **Other** An error occurred during the processing of the AIP or the user-exit program. The current save or restore task is suspended and the display from which the task was activated is redisplayed. The messages sent through the AIP are displayed on the message line of the display.
3 Reserved			4	13	This space is reserved for future use.
4 User-exit program name UEXPGM	INPUT	CHAR(10)	10	17	The name of the program to be called by the AIP.
5 Reserved			2	27	This space is reserved for future use.
6 Application name ANWID	INPUT	CHAR(7)	7	29	The name of the application to which the library to be saved or restored belongs.

Table 12 (Page 2 of 2). Interface for the Save/Restore User Exit (SAVRST)

Parameter	Use	Type	Len	Pos	Description
7 Library name LBNAM	INPUT	CHAR(10)	10	36	The name of the library that is to be saved or restored.
8 Library type LBTYP	INPUT	CHAR(1)	1	46	The identifier used in APD/400 to identify the type of the library. This can have the values: **S** Source library **O** Object library **D** Data library **J** Journal library.
9 Operation code OPRCD	INPUT	CHAR(1)	1	47	As there are four possible instances in save and restore processing where this user exit can be called, this flag is used by APD/400 to tell the AIP from what point of the save or restore process it is called: **1** Before save **2** After save **3** Before restore **4** After restore.
10 Activity flag ACTFL	OUTPUT	CHAR(1)	1	48	This flag is returned from the user exit to APD/400 and instructs APD/400 how to continue: **0** Save or restore the library LBNAM. **1** Do not save or restore the library LBNAM. This flag is evaluated only for calls prior to save or restore (OPRCD=1 and OPRCD=3). If it is 1, APD/400 skips the current library and continues with the next library to be saved or restored.
Total			**48**		

APIs

The following describes how APIs are accessed via an API server, migration of APIs between different releases of APD/400, APIs from previous releases, the interface to an API, and messages returned from an API.

API Server

You as an application developer can call certain APD/400 functions from within your application. These functions, called APIs, are accessible through a single APD/400 program, the API server, called QAFAPIPG.

The API server builds the environment in which APD/400 works, and also restores the environment in which your application works after processing of the APD/400 function.

You must pass a parameter structure to the API server program instructing it which API function you want to process, and containing all information needed by that function to perform its task.

The layout of the API interface, like that of the user-exit interface, consists of two parts:

1. The service-independent part provides communication with the API server (the name of the APD/400 service being called is passed here to the API server program). This part is described in "Calling an API."
2. The service-dependent part contains information that is needed by the individual APD/400 API to perform its task. This part is described individually for each API.

Migration

APIs described and called differently in previous releases of APD are supported by the current release and will be supported by future releases of APD/400 in parallel to the APIs described here.

An API interface, described and published, will not be changed in future releases of APD/400. API services used in an application will be the same in future APD/400 releases.

However, improvements, including extension of services of existing APD/400 APIs could be introduced in future releases. If such an API is introduced with a new APD/400 release, this API will have a new name and a new interface layout. If you want to use the extended services in your application, you will have to call the new API.

APIs from Previous Releases

The following table lists APIs that have been documented in earlier releases of APD. These APIs are supported in the current release and will be supported in future releases:

Table 13. APIs from Previous Releases

APD Version 1 Release 1 API Name	APD/400 Version 2 Release 3 API Name	Description
ADPD040	CHKAUT	Check authorization
ADPD940	SCHBATCH	Schedule batch

For performance reasons, it is recommended that applications using old APIs migrate to the new API calling convention.

Calling an API

The interface that is used for the communication between the application program and the API can be divided into two parts:

- A service-independent part is used to pass information to the API server. This part of the interface has the same layout for all APIs.
- A service-dependent part is used to pass information to each API. The layout of this part depends on the individual requirements of each API.

The service-independent part of the interface is described in Table 14 on page 64. The service-dependent part is described individually for each API.

Table 14. Service-Independent Part of the API Parameter

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	The service name. This a 10-byte name representing the purpose of the API. All service names must be in uppercase.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	If 00 is returned the service has been completed successfully. Otherwise an error has been detected. Refer to "Completion Codes" on page 64 for more details on this parameter.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
Total			**16**		

Completion Codes

A 2-byte character field. This field is always set by the API on return to the application. The following completion codes are valid:

00 **Successful Completion**. Both the API server and the service program ended without detecting any errors.

01 **Unknown Service**. The service name supplied in the parameter APINAM (service name) is invalid. Check if you have misspelled the service name.

Note: The service name must be in uppercase.

03 **No Defaults Available**. You requested to use the APD/400 defaults (you defined an asterisk (*) for one or more fields) for some of the service-dependent parameters, but the API server is not able to set the defaults. One possible reason for this error is that you requested this service from an application that is not running under APD/400 control (such as an unattached job; see case 5 in Figure 9 on page 50). In such a case, APD/400 does not know the current values for installation, application, and so on.

05 **Service Program Not Callable**. The API server could not find the program that processes the service. There are several reasons why this error can occur, such as:

- The server program object has been deleted or destroyed.
- The calling application program is not authorized to call the server program.
- The server program does not have the correct release and modification level.

Refer to the OS/400 job log to analyze the reason for this error.

06 APD/400 is currently locked, that is, another function (for example, reorganization) needs exclusive access to APD/400. The request must be tried again at a later time.

19 **Other Server Errors**. An error not described previously has occurred. In such a case, APD/400 may send messages to the program message queue of the program that called QAFAPIPG. Analyze these messages

and refer to the OS/400 job log to determine the reason for this type of error.

20 - 98 **Service-Dependent Completion Codes**. This range is reserved for the service-dependent completion codes. Refer to the description of each individual API service.

99 **Other Service-Dependent Errors**. An error that is not described in the list of service-dependent completion codes has occurred. In such a case, APD/400 may send messages to the program message queue of the program that called QAFAPIPG. Analyze these messages and refer to the OS/400 job log to determine the reason for this type of error.

Defaults for Optional Parameters

If the program calling the API is running under APD/400 control (cases 1 to 4 in Figure 9 on page 50), it is not mandatory that you specify all values required by the service. Because there is already an active APD/400 session, APD/400 could use defaults for some parameters. For example:

- Installation name
- Application name
- Task name
- Data-set name.

If one of these values is required within the service-dependent part of the API interface, the server inserts the current defaults if you specify an asterisk (*) for the first byte, and blanks (b = Hex 40) for the other bytes of the field.

For example, the batch schedule API `SCHBATCH` needs information about the installation and the application to which the batch task belongs. If the task belongs to the same installation and application, you can fill the first byte of the corresponding installation and application fields of the service-dependent part with an asterisk. APD/400 then retrieves the information for the current active task from the database, and replaces the asterisks with the corresponding values.

In the following description of each API interface, the parameters that are optional (for which APD/400 may provide a default) are marked with an asterisk in the first character of the `Use` column. For example, the installation name (`INSID`) field of the `SCHBATCH` service is optional and marked as `*INPUT` in the `Use` column of the interface description table. A required parameter would be marked as `INPUT` (without the leading asterisk).

The `Description` column for each optional parameter contains an explanation of the default.

Note: You cannot use the APD/400 defaults for one of the service-dependent parameters from an application that is not running under control of APD/400 (for example, an unattached job; see case 5 in Figure 9 on page 50). In such a case, APD/400 does not know the current values for installation, application, and so on. You must provide existing values for all the service-dependent parameters even if they are defined as replaceable by defaults.

You can use this feature for all or for a subset of the parameters that are defined as replaceable by defaults.

Refer to the description of the individual services for a more detailed description of parameter defaults.

Note: If work with defaults in APIs is switched off, no information from the active APD/400 session will be provided for this application.

Messages

All APIs return messages to the calling application when an error has been detected that cannot be matched to the completion codes 01 through 18 (for service-independent errors), and 20 through 98 (for service-dependent errors). The messages are sent to the program message queue of the program that called QAFAPIPG. If the completion code 19 or 99 is returned, receive these messages to your program to analyze the problem.

API Descriptions

The following are the APIs supplied with APD/400.

ADDADTE Add Audit File Entry

This API is used to add records to the audit file QAAFAUDT0, to save information on the current job that a user wants to keep. The user can trace the job by calling this API at each step where a problem could occur, or at specific events. All values are optional. The user can decide which information is necessary in each case.

Interface Description

Table 15 (Page 1 of 2). Parameters for the Add Audit File Entry API (ADDADTE)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	ADDADTE The service name of the Add Audit File Entry API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to “Completion Codes” on page 64 for more details on completion codes. **99** An error during a file operation on file QAAFAUDT0 has occurred. See the messages.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	*INPUT	CHAR(3)	3	17	The name of the installation to which the task defined in TSKID belongs. **Default:** The name of the installation to which the currently active task belongs.
5 Application name ANWID	*INPUT	CHAR(7)	7	20	The name of the application to which the task to be processed belongs. **Default:** The name of the application to which the currently active task belongs.

Table 15 (Page 2 of 2). Parameters for the Add Audit File Entry API (ADDADTE)

Parameter	Use	Type	Len	Pos	Description
6 Task name TSKID	INPUT	CHAR(10)	10	27	In this field the name of the task or program that is processed is stored.
7 User name USRID	*INPUT	CHAR(10)	10	27	The name of the user processing the task. **Default:** The user profile of the user to which the currently active job belongs.
8 Data-set name FIRNR	*INPUT	CHAR(4)	4	47	The name of the data set to which the task to be processed belongs. **Default:** The name of the currently active data set of the currently active application. Leave this parameter blank if the application does not use data sets.
9 Job name JOBID	*INPUT	CHAR(10)	10	51	The name of the current job. This value can be found with RTVJOBA. **Default:** The job name of the current job.
10 Job number JOBNR	*INPUT	CHAR(6)	6	61	The job number of the current job. This value can be found with RTVJOBA. **Default:** The job number of the current job.
11 Processing type EXTYP	INPUT	CHAR(1)	1	67	Processing type: B for batch or I for interactive.
12 Event type EVNTP	INPUT	CHAR(7)	7	68	Describes the event that is to be stored, such as START, END, CANCEL, and so on.
Total			**74**		

Note: None of the fields are checked for validity.

Example

In the following example, the ADDADTE API is used to log the processing (start and end) of a subtask in a COBOL program:

```
---+-*- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5 ---+--- 6 ---+--- 7
      *    Define the structure for the interface of the API.
      *    Call the structure ADDADTE to be consistent with the rest of
      *    the example.
      .
      .
       PROCEDURE DIVISION.
           SAMPLE-PROGRAM.

      *    // Initialize the parameters for the API server.
           MOVE "ADDADTE"   TO    APINAM  IN ADDADTE
           MOVE "00"        TO    APICCD  IN ADDADTE
           MOVE SPACES      TO    APIFTU  IN ADDADTE
           MOVE "   "       TO    INSID   IN ADDADTE
           MOVE "APPL123    TO    ANWID   IN ADDADTE
           MOVE "*"         TO    USRID   IN ADDADTE
           MOVE "0001"      TO    FIRNR   IN ADDADTE
           MOVE "*"         TO    JOBID   IN ADDADTE
           MOVE "*"         TO    JOBNR   IN ADDADTE
           MOVE "B"         TO    EXTYP   IN ADDADTE

      *    // Process procedure PROC01 and log start and end
      *    // using the API ADDADTE.
           MOVE "PROC01"    TO    TSKID   IN ADDADTE
           MOVE "START"     TO    EVNTP   IN ADDADTE
           CALL "QAFAPIPG"  USING ADDADTE IN QAFAPIPG
           PERFORM PROC01
           MOVE "END"       TO    EVNTP   IN ADDADTE
           CALL "QAFAPIPG"  USING ADDADTE IN QAFAPIPG

      *    // Process procedure PROC02 and log start and end
      *    // using the API ADDADTE.
           MOVE "PROC02"    TO    TSKID   IN ADDADTE
           MOVE "START"     TO    EVNTP   IN ADDADTE
           CALL "QAFAPIPG"  USING ADDADTE IN QAFAPIPG
           PERFORM PROC02
           MOVE "END"       TO    EVNTP   IN ADDADTE
           CALL "QAFAPIPG"  USING ADDADTE IN QAFAPIPG
```

The entries in the Audit file are built according to the passed data, including the time and the date when the event happened (02/05/1993 08:00:00), and the current job IDs (JOB1 and 123456).

The fields in the Audit file are inserted as follows:

```
QAAFAUDT0      1.            2.            3.            4.
field          Record        Record        Record        Record
name
-------------------------------------------------------------------
USRID          'USER1'       'USER1'       'USER1'       'USER1'
JOBID          'JOB1'        'JOB1'        'JOB1'        'JOB1'
JOBNR          '123456'      '123456'      '123456'      '123456'
EVTDS          x'19930205'   x'19930205'   x'19930205'   x'19930205'
EVTTS          x'080000'     x'080000'     x'080000'     x'080000'
EVTDA          '19930205'    '19930205'    '19930205'    '19930205'
EVTTA          '080000'      '080000'      '080000'      '080000'
EVTDD          '1993-02-05'  '1993-02-05'  '1993-02-05'  '1993-02-05'
EVTTD          '08:00:00'    '08:00:00'    '08:00:00'    '08:00:00'
INSID          '   '         '   '         '   '         '   '
ANWID          'APPL123'     'APPL123'     'APPL123'     'APPL123'
MTASK          'PROC01'      'PROC01'      'PROC02'      'PROC02'
FIRNR          '0001'        '0001'        '0001'        '0001'
RETCD
BJSNR          0             0             0             0
RSSNR          0             0             0             0
EVNTP          'START'       'END'         'START'       'END'
EXTYP          'B'           'B'           'B'           'B'
```

Note: The date format in the EVTDD field depends on the setting of the APD/400 parameter APD_DATE_REPRESENTATION at the time the ADDADTE API is called.

CHGAPPD Change Application Definitions

This API updates an application according to changes created by using the Compare Application Definitions (CMPAPPD) API. The changes (inserts, deletes, or updates) are applied to the current version that is stored in the APD/400 data repository in library QUSRSYS.

Note: In the following, before-image refers to the previous version, and after-image to the new version.

Definitions are inserted as follows. If the record:

- Does not already exist, it is inserted.
- Already exists and has the same content as the after-image, it is ignored.
- Already exists and has a content different from the after-image, it is replaced with the after-image and the current values of the deviating fields are printed in the error report.

Definitions are deleted as follows. If the record:

- Exists and has the same content as the before-image, it is deleted.
- Does not exist, it is ignored.
- Exists and has a content different from the before-image, it is deleted and the current values of the deviating fields are printed in the error report.

Definitions are changed as follows. If the record:

- Exists and has the same content as the before-image, it is replaced with the after-image.
- Exists and has the same content as the after-image, it is ignored.

- Exists and has a content different from both the before-image and the after-image, it is replaced with the after-image, and the current values of the fields different from the before-image and after-image are printed in the error report.

  Note: The comparison is done field-by-field. No error is reported for fields of which the before-image and after-image are identical. In this case, the current content of the field is retained even though other fields in the record may be updated.

- If the record does not exist, the after-image is inserted and a corresponding error is printed.

The AIP is always replaced.

Figure 10 shows a sample warning report.

```
5716PD1 V3R6M0  950430                IBM AS/400 Application Program Driver/400                04/30/95  10:30:16    Page     1
Change Application Description (Warning report)              DT       0000ADT

Record key            Field  Description                     Replaced field contents

Application file
DT                    ANWTX  Application text                APDC Development Tools       (PTR Tool)
DT                    AUDTF  Audit flag                      1
```

Figure 10. Sample Warning Report

“Sample Scenarios” on page 45 shows how this API can be used.

Interface Description

Table 16 (Page 1 of 2). Parameters for the Change Application Definitions API (CHGAPPD)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	CHGAPPD The service name of the Change Application Definitions API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **20** The specified installation does not exist. **21** The specified application does not exist in either the current APD/400 tables or in the input library. **22** You do not have authority for the application. You must be the administrator of the application (the APD/400 administrator can authorize you to use the Administer Applications function (ADMAPL)), or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **23** The AIP does not exist in the input library. **25** The input library does not exist. **26** The input library contains incorrect application definitions. **27** The APD/400 version of the application definitions in the input library could not be determined. **28** The operation code is not valid. **29** The specified application is currently in use. All users must suspend use of the application while changes are being applied. **58** All processing has completed successfully, but a warning report has been printed listing the changes that have been overwritten. **99** Other errors. See the job log.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	INPUT	CHAR(3)	3	17	The name of the installation containing the application for which the definitions are to be changed. **Default:** The name of the installation to which the currently active task belongs.
5 Application name ANWID	INPUT	CHAR(7)	7	20	The name of the application for which the definitions are to be changed. **Default:** The name of the application to which the currently active task belongs.
6 Definition library DLIBC	INPUT	CHAR(10)	10	27	The name of the library containing the changes to the application definitions.

Table 16 (Page 2 of 2). Parameters for the Change Application Definitions API (CHGAPPD)

Parameter	Use	Type	Len	Pos	Description
7 Operation code OPRCD	INPUT	CHAR(1)	1	37	This API can be used in two different modes depending on whether you want updates to the application definitions to be applied: **0** A warning report listing the user changes that would be destroyed is written but no actual updates are made to the current application definitions. **1** The current application definitions are updated and a report listing the user changes that have been destroyed is written. A report is not written (independent of the operation code) if there are no replacement definitions.
Total			**37**		

CHGDST Change Data Set

This API is used to change the current data set of an application from within an application program. It has the same effect as the `Select Data Sets` function (see *IBM Application Program Driver/400 Version 3: User's Guide*). The change is valid for the user and job. The data set selected is used for the current job and as the default after the next sign-on.

Interface Description

Table 17 (Page 1 of 2). Parameters for the Change Data-Set API (CHGDST)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	CHGDST The service name of the Change Data-Set API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **99** Other errors.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	*INPUT	CHAR(3)	3	17	The name of the installation to which the data set to be changed belongs. **Default:** The name of the installation to which the currently active task belongs.
5 Application name ANWID	*INPUT	CHAR(7)	7	20	The name of the application to which the data set to be changed belongs. **Default:** The name of the application to which the currently active task belongs.
6 Data-set name FIRNR	INPUT	CHAR(4)	4	27	The name of the data set that should be used as the new active data set for the current job.

Table 17 (Page 2 of 2). Parameters for the Change Data-Set API (CHGDST)

Parameter	Use	Type	Len	Pos	Description
7 Return code RETCDE	OUTPUT	CHAR(1)	1	31	**0** Data set has been changed. **1** Data set does not exist. A data set with the given installation, application, and data-set name does not exist. **2** User not authorized to use the data set. This is returned if the data set is secured with an APD/400 authorization list and the current user does not have an authorization level of 1 or greater. **3** Data set is locked. Another job is processing a task that requires exclusive access to the data set (an exclusion of type 2 has been defined for the task).
Total			**31**		

Note: The setting Work with Data Sets has no influence on this API.

CHKAUT Check Authorization

This API is used to retrieve the authorization level for a given user from an APD/400 authorization list.

Authorization lists can be used for APD/400 objects such as tasks and data sets. This API can be used to provide APD/400 authorization lists for objects of an application, so that it is not necessary to develop a new authorization concept.

Interface Description

Table 18 (Page 1 of 2). Parameters for the Check Authorization API (CHKAUT)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	CHKAUT The service name of the Check Authorization API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to “Completion Codes” on page 64 for more details on completion codes. **20** The authorization list name (field BRFKT) or user ID (field USRID) was blank when the API was called. No authorization level could be determined. **99** An error occurred during the authorization checking. (This return code occurs, for example, if the authorization list name is unknown.)
3 Future use APIFTU			4	13	This field is reserved for future use. However, you should fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	*INPUT	CHAR(3)	3	17	The name of the installation to which the authorization list to be checked belongs. **Default:** The installation name of the currently active task.

Table 18 (Page 2 of 2). Parameters for the Check Authorization API (CHKAUT)

Parameter	Use	Type	Len	Pos	Description
5 Application name ANWID	*INPUT	CHAR(7)	7	20	The name of the application to which the authorization list to be checked belongs. **Default:** The application name of the currently active task.
6 User name USRID	*INPUT	CHAR(10)	10	27	The name of the user processing the task. **Default:** The user profile of the user who owns the current job.
7 Authorization list name BRFKT	INPUT	CHAR(10)	10	37	The name of the authorization list to be checked.
8 Authorization level BRSTF	OUTPUT	CHAR(1)	1	47	The 1-character (0 to 9) authorization level retrieved from the authorization list.
Total			**47**		

Note: The setting Use Authorization Checking has no influence on API CHKAUT.

CHKEXC Check Exclusion

This API service can be called to use the APD/400 exclusion control from within your application program. With this service you can check, set, and reset an exclusion for an exclusion list.

Note: If you set an exclusion for an exclusion list with this API, you must reset it when the function that required the exclusive access ends. Otherwise, the exclusion record remains in the APD/400 database and may lock other functions. In such a case, you must process the database reorganization with at least level 2. See “Reorganization” in the APD/400 *IBM Application Program Driver/400 Version 3: Administrator's Guide*.

Interface Description

Table 19 (Page 1 of 3). Parameters for the Check Exclusion API (CHKEXC)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	CHKEXC The service name of the Check Exclusion API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion **01 - 19** Refer to “Completion Codes” on page 64 for more details on completion codes. **20** The field Operation Code (OPRCD) contained an invalid value. The value of RETCDE is undefined. **23** Data set not found. **24** Not authorized for data set. **99** Other errors. See the job log.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.

Table 19 (Page 2 of 3). Parameters for the Check Exclusion API (CHKEXC)

Parameter	Use	Type	Len	Pos	Description
4 Installation name INSID	*INPUT	CHAR(3)	3	17	The name of the installation to which the exclusion list to be allocated or deallocated belongs. **Default:** The name of the installation to which the currently active task belongs.
5 Application name ANWID	*INPUT	CHAR(7)	7	20	The name of the application to which the exclusion list to be allocated or deallocated belongs. **Default:** The name of the application to which the currently active task belongs.
6 Exclusion list name PGMGR	INPUT	CHAR(10)	10	27	The name of the exclusion list to be allocated or deallocated.
7 Operation code OPRCD	INPUT	CHAR(1)	1	37	This API can be used in two different modes: one to check and set an exclusion, and one to release an exclusion. The mode is controlled by the operation code as follows: **1** Check and set an exclusion for the exclusion list. **2** Release an exclusion for the exclusion list.
8 Return code RETCDE	OUTPUT	CHAR(1)	1	38	The API passes a return code back to the caller program as follows: **0** The operation was successful and the exclusion could be set or released. **1** For OPRCD = 1, the exclusion could not be set, because it is excluded by another task. For OPRCD = 2, the exclusion has not been released, because no activity record could be found for this exclusion list. If the completion code (APICCD) does not contain 00, the return code RETCDE is undefined.
9 Data set Id FIRNR	*INPUT	CHAR(4)	4	39	The ID of the data set, for which the exclusion list should be allocated or deallocated. **Default:**The current active data set. (this one which was selected at last is the current data set).

Table 19 (Page 3 of 3). Parameters for the Check Exclusion API (CHKEXC)

Parameter	Use	Type	Len	Pos	Description
10 Return message variable RETMSG	OUTPUT	CHAR(48)	48	43	A record format which contains the • application-id (char7) • alias-name (char7) • task-id (char10) • exclusion list (char10) • user-id (char10) • data set-id (char4) which causes the exclusion. This field is filled, if an exclusion is identified. If an active task is identified, • alias-name • task-id • user-id • data-set will be returned. If only an exclusion is identfied without finding the corresponding active task (e.g., if the exclusion was set using the API CHKEXC), • application-id • exclusion list • data set will be returned.
Total			**90**		

Example 1

In this example, a task T1 is processing under APD/400. The program that belongs to T1 is an online program that displays a list of items, where the user can select one of the items for processing by typing an option code against the item. The options are 2=Change, 4=Delete, 5=Display, and 6=Print. This is a typical WRKxxx list display as often used by OS/400.

No exclusion control has been defined on the task level, because it can be decided only on the subtask level whether processing is excluded or not. Having an exclusion on the task level that avoids double invocation of T1 would be too restrictive.

The solution is to use a set of subtasks named T1_2, T1_4, T1_5, and T1_6 that correspond to the processing performed when options 2, 4, 5, or 6 are selected. The following table provides an overview of the tasks and subtasks that belong to the sample program:

Table 20. Tasks and Subtasks

Task	Option	Subtask	Exclusion List
T1	2	T1_2	T1_UPDATE
T1	4	T1_4	T1_UPDATE
T1	5	T1_5	
T1	6	T1_6	

Note: Subtasks T1_2 through T1_6 need not necessarily be defined as tasks within APD/400, as the API requires only the name of the exclusion list to which the task belongs.

In this example, the change and delete processes require exclusive access to the database of the application. Therefore, within APD/400 an exclusion has been defined as follows:

```
T1_UPDATE ◄————————► T1_UPDATE
```

This exclusion definition guarantees that only one task belonging to the exclusion list can be active at a given point in time.

Note: The setting Use Exclusion Checking has no influence on `API CHKEXC`.

Code for the example is as follows:

```
---+-*- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5 ---+--- 6 ---+--- 7
     *    Define the structure for the interface of the API.
     *    Call the structure CHKEXC to be consistent with the rest of
     *    the example.

     *    // The interface for the call to QAFAPIPG is
     *    // initialized.  The installation ID (INSID) and
     *    // application ID (ANWID) must not be explicitly
     *    // defined as they are the same as for the current
     *    // active task T1.  Therefore, an asterisk (*) is used
     *    // to tell the APD/400 API server to insert the defaults.
          MOVE "CHKEXC"    TO    APINAM  IN CHKEXC
          MOVE "00"        TO    APICCD  IN CHKEXC
          MOVE SPACES      TO    APIFTU  IN CHKEXC
          MOVE "*"         TO    INSID   IN CHKEXC
          MOVE "*"         TO    ANWID   IN CHKEXC

          PERFORM UNTIL F03-PRESSED
     *      // Write and read the display file that displays
     *      // the WRKxxx panel.
            PERFORM GET-OPTION-FROM-PANEL
            IF NOT-F03-PRESSED
            THEN
              EVALUATE TRUE
              WHEN OPTION = "2" OR OPTION = "4"
     *          // Option 2 and 4 (subtask T1_2 and T1_4) belong
     *          // to the T1_UPDATE exclusion group.
                MOVE "T1_UPDATE"   TO    PGMGR  IN CHKEXC
                MOVE "1"           TO    OPRCD  IN CHKEXC
                CALL "QAFAPIPG"    USING CHKEXC IN QAFAPIPG
                IF RETCDE IN CHKEXC = "0"
                THEN
     *            // No exclusion, processing may continue.
                  EVALUATE OPTION
                  WHEN "2" PERFORM PROC02
                  WHEN "4" PERFORM PROC04
                  END-EVALUATE

     *            // Deallocate the exclusion group when processing
     *            // has finished.
                  MOVE "2"           TO    OPRCD  IN CHKEXC
                  CALL "QAFAPIPG"    USING CHKEXC IN QAFAPIPG
                ELSE
     *             // T1_UPDATE is currently excluded by different
     *             // processing. Send a message and redisplay the
     *             // panel.
                END-IF

     *        // Options 5 and 6 (display and print)
     *        // do a read access on the database.
     *        // No exclusion checking is necessary.
              WHEN OPTION = "5" PERFORM PROC05
              WHEN OPTION = "6" PERFORM PROC06
              WHEN OTHER        CONTINUE
              END-EVALUATE
            END-IF
          END-PERFORM
```

Example 2

This API can also be used to exclude a task, all tasks of an application, or all tasks within APD/400 without calling all excluding tasks. For example, an application can be excluded during installation of a database upgrade, even though the upgrade program does not run under APD/400 control.

CMPAPPD Compare Application Definitions

This API compares the new and previous definitions of the specified application, creating a set of changed definitions. The records in each file contain a flag indicating whether the record is to be inserted, deleted, or updated (with a before-image and an after-image). The AIP is always copied.

"Sample Scenarios" on page 45 describes how this API can be used.

Interface Description

Table 21 (Page 1 of 2). Parameters for the Compare Application Definitions API (CMPAPPD)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	CMPAPPD The service name of the Compare Application Definitions API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **21** The specified application does not exist in at least one of the input libraries. **22** You do not have authority for the application. You must be the administrator of the application as defined in one of the input libraries, or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **23** The AIP does not exist in at least one of the input libraries. **24** The library already exists. **25** At least one of the input libraries does not exist. **26** At least one of the input libraries contains incorrect application definitions. **27** The APD/400 version of the application definitions in at least one of the input libraries could not be determined. **99** Other errors. See the job log.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Application name ANWID	INPUT	CHAR(7)	7	17	The name of the application for which the definitions are to be compared. **Default:** The name of the installation to which the currently active task belongs.
5 Definition library (previous version) DLIBP	INPUT	CHAR(10)	10	24	The name of the library containing the definitions of the previous version of the application.

Table 21 (Page 2 of 2). Parameters for the Compare Application Definitions API (CMPAPPD)

Parameter	Use	Type	Len	Pos	Description
6 Definition library (new version) DLIBN	INPUT	CHAR(10)	10	34	The name of the library containing the definitions of the new version of the application.
7 Definition library (changes) DLIBC	INPUT	CHAR(10)	10	44	The name of the library to receive the changes to the application definitions. It should not exist when the API is invoked.
Total			**53**		

DLTAPPD Delete Application Definitions

This API deletes the APD/400 part (application definitions and AIP) of an application. It performs a function similar to `Delete Applications`.

Note: Only the APD/400 part is deleted. Application objects such as libraries, folders, and user profiles are not changed.

"Sample Scenarios" on page 45 describes how this API can be used.

Interface Description

Table 22 (Page 1 of 2). Parameters for the Delete Application Definitions API (DLTAPPD)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	DLTAPPD The service name of the Delete Application Definitions API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **20** The specified installation does not exist. **21** The specified application does not exist in the specified installation. **22** You do not have authority for the application. You must be the administrator of the application (the APD/400 administrator can authorize you to use the `Administer Applications` function (ADMAPL)), or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **29** The specified application is currently in use. All users must suspend use of the application while changes are being deleted. **34** The specified application cannot be deleted because it is a prerequisite for another application. **35** The application APD cannot be deleted from the standard installation. **99** Other errors. See the job log.

Table 22 (Page 2 of 2). Parameters for the Delete Application Definitions API (DLTAPPD)

Parameter	Use	Type	Len	Pos	Description
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	INPUT	CHAR(3)	3	17	The name of the installation containing the application for which the definitions are to be deleted. **Default:** The name of the installation to which the currently active task belongs.
5 Application name ANWID	INPUT	CHAR(7)	7	20	The name of the application for which the definitions are to be deleted.
Total			**26**		

DSPINSAPP Display Installed Applications

This API allows the developer or user to display selected installations, applications, and data sets in a specified output file.

"Sample Scenarios" on page 45 describes how this API can be used.

Interface Description

Table 23 (Page 1 of 2). Parameters for the Display Installed Applications API (DSPINSAPP)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	DSPINSAPP The service name of the Display Installed Applications API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **22** You do not have authority to display installed applications. You must be an application administrator (the APD/400 administrator can authorize you to use the Administer Applications function (ADMAPL)), or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **31** At least one of the parameters for the output file is not valid, or the file or member already exists and *NEWFILE or *NEWMBR was specified. **99** Other errors. See the job log.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	*INPUT	CHAR(4)	4	17	The name of the installation. *ALL means display all installations. A generic name can also be used. **Default:** The name of the installation to which the currently active task belongs.

Table 23 (Page 2 of 2). Parameters for the Display Installed Applications API (DSPINSAPP)

Parameter	Use	Type	Len	Pos	Description
5 Application name ANWID	*INPUT	CHAR(7)	7	21	The (internal) name of the application. *ALL means display all installations. A generic name can also be used. **Default:** The name of the installation to which the currently active task belongs.
6 Data-set name FIRNR	*INPUT	CHAR(4)	4	28	The name of the data set. *ALL means display all data sets. A generic name can also be used. **Default:** The name of the currently active data set in the application to which the currently active task belongs.
7 File name OUTFILE	INPUT	CHAR(10)	10	32	The name of the database file that receives the output. If the database file does not exist, the system creates it in the specified library.
8 Library name LIB	INPUT	CHAR(10)	10	42	The name of the library where the database file is located, or *CURLIB.
9 Member name MBR	INPUT	CHAR(10)	10	52	The file member to receive the output, or *FIRST.
10 Output option OPTION	INPUT	CHAR(8)	8	62	The possible values for output options are: ***NEWFILE** The output is written to a new database file. ***RPLFILE** The output deletes the old file if it exists, and creates a new database file. ***NEWMBR** The output is added as a new member. ***RPLMBR** The existing member is cleared and the output is added. ***ADDMBR** The output is added to the end of an existing member.
Total			**69**		

The following is the layout of the outfile produced by this API:

Table 24 (Page 1 of 2). Layout of Outfile for Display Installed Applications

Field	Pos	Len	Description
INSID	1	3	Installation ID The ID of the installation to which the application belongs.
INSTX	4	40	Installation description The description of the installation to which the application belongs.
BRFKT	44	10	Authorization list ID of installation The ID of the authorization list that is used to secure the installation. The authorization list belongs to the application APD in the default installation (ḃḃḃ). The ID of the authorization list is INST_xxx, where xxx is replaced by the installation ID.

Table 24 (Page 2 of 2). Layout of Outfile for Display Installed Applications

Field	Pos	Len	Description
ANWID	54	7	Application ID The (internal) ID of the application.
ALIAS	61	7	Alias The (external) ID of the application.
FTRCD	68	4	Future use Currently not used.
VRSST	72	2	Version The version of the application.
RLSST	74	2	Release The release of the application.
MDLVL	76	4	Modification level The modification level of the application.
ANADM	80	10	Application administrator The user profile name of the application administrator.
ANWTX	90	40	Application text The descriptive text for the application.
AUDTF	130	1	Audit flag Indicates whether activities of the application are audited or not. Values are: **0** Do not audit activities **1** Audit activities.
MNUBL	131	1	Blank line for missing menu options flag Indicates whether blank lines are inserted between menu options of the application if the option numbers are not in a sequence. Values are: **0** Do not insert blank line **1** Insert blank line.
MHFID	132	10	Menu headings format ID The ID of the menu heading format to be used for menus of the application. This menu heading format is used only if one is not specified on the task level.
MNUFT	142	1	Single/double column menu format flag Indicates whether the single or double column layout is used for menus of the application. Values are: **1** Single column **2** Double column.
FIRNR	143	4	Data set ID The ID of a data set for the application.
FIRNM	147	40	Data set description The description of a data set for the application.
BRFKT01	187	10	Authorization list ID of data set The ID of the authorization list used to secure the data set.

EXTAPPD Extract Application Definitions

This API extracts the APD/400 part (application definitions and AIP) of the given application from the APD/400 repository and stores it in a specified library. The procedure is basically the same as that performed by option `3` (Copy to library QAPDIAHDR) of the `Administer Applications (Developer)` function (ADMAPP).

"Sample Scenarios" on page 45 describes how this API can be used.

Interface Description

Table 25 (Page 1 of 2). Parameters for the Extract Application Definitions API (EXTAPPD)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	EXPAPPD The service name of the Extract Application Definitions API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **20** The specified installation does not exist. **21** The specified application does not exist. **22** You do not have authority for the application. You must be the administrator of the application (the APD/400 administrator can authorize you to use the `Administer Applications` function (ADMAPL)), or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **23** The AIP does not exist. **24** The library already exists. **36** No menu selection was found for this application in the APD/MAIN menu. The extraction of the application definitions was completed despite this minor error. If the resulting definitions are installed, no menu selection is inserted into the APD/MAIN menu. **57** More than one menu selection was found for this application in the APD/MAIN menu. The extraction of the application definitions was completed despite this minor error. If the resulting definitions are installed, only the first menu selection is inserted into the APD/MAIN menu. **99** Other errors. See the job log.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (`b` = Hex 40) before you call the server.
4 Installation name INSID	INPUT	CHAR(3)	3	17	The name of the installation containing the application for which the definitions are to be extracted. **Default:** The name of the installation to which the currently active task belongs.

Table 25 (Page 2 of 2). Parameters for the Extract Application Definitions API (EXTAPPD)

Parameter	Use	Type	Len	Pos	Description
5 Application name ANWID	INPUT	CHAR(7)	7	20	The name of the application for which the definitions are to be extracted. **Default:** The name of the application to which the currently active task belongs.
6 Definition library DLIBN	INPUT	CHAR(10)	10	27	The name of the library to receive the extracted application definitions. It should not exist when the API is evoked.
Total			**36**		

INSAPPD Install Application Definitions

This API installs the APD/400 part (application definitions and AIP) of the given application. It performs a function similar to `Install Applications`.

Note: Only the APD/400 part of the application is installed. No user profiles, libraries, or folders are installed or created.

"Sample Scenarios" on page 45 describes how this API can be used.

Interface Description

Table 26 (Page 1 of 3). Parameters for the Install Application Definitions API (INSAPPD)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	INSAPPD The service name of the Install Application Definitions API.

Table 26 (Page 2 of 3). Parameters for the Install Application Definitions API (INSAPPD)

Parameter	Use	Type	Len	Pos	Description
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **20** The specified installation does not exist. **21** The specified application does not exist in the input library. **22** You do not have authority for the application. You must be the administrator of the application (the APD/400 administrator can authorize you to use the Administer Applications function (ADMAPL)), or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **23** The AIP does not exist in the input library. **25** The input library does not exist. **26** The input library contains incorrect application definitions. **27** The APD/400 version of the application definitions in the input library could not be determined. **29** The specified application is currently excluded by other processing. **32** The specified application already exists in the specified installation. **33** The specified application cannot be installed because a prerequisite application has not yet been installed. **42** The data-set description is blank. **43** The data-set name contains characters that are not valid. **46** The save object name is incorrect. **99** Other errors. See the job log.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	INPUT	CHAR(3)	3	17	The name of the installation that contains the application for which the definitions are to be installed. **Default:** The name of the installation to which the currently active task belongs.
5 Application name ANWID	INPUT	CHAR(7)	7	20	The name of the application for which the definitions are to be installed.
6 Data-set name FIRNR	INPUT	CHAR(4)	4	27	The name of the data set. This parameter is required only when the name of the data set is used in forming library names (replacement variables &Dn appear in the library name templates). Otherwise, it is blank. Valid characters are A-Z and 0-9. Characters used in the library name must not be blank.

Table 26 (Page 3 of 3). Parameters for the Install Application Definitions API (INSAPPD)

Parameter	Use	Type	Len	Pos	Description
7 Data-set description FIRNM	INPUT	CHAR(40)	40	31	The description of the data set. This parameter is required only when the name of the data set is used in forming library names (replacement variables &Dn appear in the library name templates). Otherwise, it is blank.
8 Definition library DLIBN	INPUT	CHAR(10)	10	71	The name of the library containing the application definitions.
Total			**80**		

SCHBATCH Schedule a Batch Task

This API is used to enable a user to schedule a batch task through APD/400. The service program checks whether the current user is authorized to perform the task and to use the batch environment that is defined for the task.

Note: Schedule a Batch Job (where you can override the schedule time, batch environment, and so on) does not display for batch tasks scheduled using this API. It displays only when you schedule the batch task from a menu or using an expert code.

Interface Description

Table 27 (Page 1 of 2). Parameters for the Schedule Batch Task API (SCHBATCH)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	SCHBATCH The service name of the Schedule a Batch task API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to “Completion Codes” on page 64 for more details on completion codes. **20** - The task type is not program or command (process lists are not supported). - The type of processing is not batch. **21** The task does not exist. **22** The user is not authorized for the task. **23** The data set does not exist. **24** The user is not authorized for the data set. **26** The restart flag is not valid. **27** The audit flag is not valid. **28** The date or time is not valid. **99** Other errors. The job is not scheduled. See other messages.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.

Table 27 (Page 2 of 2). Parameters for the Schedule Batch Task API (SCHBATCH)

Parameter	Use	Type	Len	Pos	Description
4 Installation name INSID	*INPUT	CHAR(3)	3	17	The name of the installation to which the currently active task belongs. **Default:** The name of the installation to which the currently active task belongs.
5 Application name ANWID	*INPUT	CHAR(7)	7	20	The name of the application to which the task to be processed belongs. **Default:** The name of the application to which the currently active task belongs.
6 Task name TSKID	INPUT	CHAR(10)	10	27	The name of the task to be processed. **Default:** The name of the currently active task.
7 Data-set name FIRNR	*INPUT	CHAR(4)	4	37	The name of the data set to be used by the batch job. **Default:** The name of the data set to which the currently active task belongs. This parameter should be left blank if the application does not use data sets.
8 Restart flag RSTFL	*INPUT	CHAR(1)	1	41	The following values are allowed for the restart flag: **0** No restart **1** Normal restartable **2** Mandatory restart. **Default:** The restart flag as it has been defined for the task.
9 Audit flag AUDTF	*INPUT	CHAR(1)	1	42	The following values are allowed for the audit flag: **0** The task is not audited **1** The task is audited. **Default:** The audit flag as it has been defined for the task.
10 Time stamp DATTIM	*INPUT	CHAR(14)	14	43	Date and time when the job is to be processed in the format YYYYMMDDHHMMSS. **Default:** The current date and time is used, so the task is processed immediately.
11 Task parameter MPPRT	*INPUT	CHAR(512)	512	57	The task parameter that is required by the application program. **Default:** The parameter as it has been defined for the task. You must define an asterisk for byte 1 of the parameter, and all spaces (b = Hex 40) for bytes 2 through 512 of the task parameter if you want to instruct APD/400 to insert the default.
Total			**568**		

Example

In this example, a batch task is scheduled from an application program to process under the control of APD/400. The batch program prints database records from files. It expects the name of the file and the lower and upper limits for the records to be printed in the task parameter `MPPRT`. The online program prompts the user to enter the required parameters and then calls the `SCHBATCH` API to allow the task to process under control of APD/400.

This example also shows you how to instruct APD/400 to insert the defaults to the interface:

- The task to be scheduled belongs to the same installation, application, and data set as the task that processes the program described.
- For the audit and restart flags, the values from the task definition are used.
- The date and time that the task is processed is set to the current date and time.

Therefore, asterisks (*) are used as the first byte for these parameters.

Code for the example is as follows:

```
---+-*- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5 ---+--- 6 ---+--- 7
     *    Define the structure for the interface of the API.
     *    Call the structure SCHBATCH to be consistent with the rest of
     *    the example.
     .
     .
     .
      PROCEDURE DIVISION.
          SAMPLE-PROGRAM.

     *    // Initialize the parameters for the API server.
          MOVE "SCHBATCH"  TO    APINAM  IN SCHBATCH
          MOVE "00"        TO    APICCD  IN SCHBATCH
          MOVE SPACES      TO    APIFTU  IN SCHBATCH
          MOVE "*"         TO    INSID   IN SCHBATCH
          MOVE "*"         TO    ANWID   IN SCHBATCH
          MOVE "PRINTFILE" TO    TSKID   IN SCHBATCH
          MOVE "*"         TO    FIRNR   IN SCHBATCH
          MOVE "*"         TO    RSTFL   IN SCHBATCH
          MOVE "*"         TO    AUDTF   IN SCHBATCH
          MOVE "*"         TO    DATTIM  IN SCHBATCH

     *    // A panel is displayed where the user can define the
     *    // filename and the lower and upper limits for the
     *    // records to be printed.
          PERFORM GET-PARAMETER-FROM-USER
          STRING
              FILENAME IN DISPLAY-FILE-RECORD DELIMITED BY SIZE
              LOWER    IN DISPLAY-FILE-RECORD DELIMITED BY SIZE
              UPPER    IN DISPLAY-FILE-RECORD DELIMITED BY SIZE
            INTO MPPRT IN SCHBATCH
          END-STRING

          CALL "QAFAPIPG"  USING SCHBATCH IN QAFAPIPG
```

SETRST Set Restart Code

To determine the restartability of a task, APD/400 normally uses the restart flag as defined by the developer or user as a default.

This API is used to dynamically overwrite the restart flag of a task that is processing. This could be necessary if a long-running task (especially a task running in batch) needs to have different restartability options within different steps of processing. If, for example, the task abends within a complex update of a database file, the task must be restarted (mandatory restart), but when the task is creating a report this is not necessary.

Therefore, you can use this API to change the restartability of a task according to your requirements during runtime.

Interface Description

Table 28 (Page 1 of 2). Parameters for the Set Restart Code API (SETRST)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	SETRST The service name of the Set Restart Code API.

Table 28 (Page 2 of 2). Parameters for the Set Restart Code API (SETRST)

Parameter	Use	Type	Len	Pos	Description
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **21** One of the following errors has occurred: • The restart flag is not valid. • No job entries in ADJOBS0, ADSCDL0, or ADSTCK0. **99** Other errors.
3 Future use APIFTU		CHAR(4)	4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Restart flag RSTFL	INPUT	CHAR(1)	1	17	The following values are allowed for the restart flag: **0** Not restartable **1** Restartable **2** Mandatory restartable.
Total			**17**		

Example

The following is a sample program that uses the SETRST API:

```
---+-*- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5 ---+--- 6 ---+--- 7
     *    Copy the interface record for the SETRST API call
     *    to the program.
     *    Define the structure for the interface of the API.
     *    Call the structure SETRST to be consistent with the rest of
     *    the example.
     .
     .
     .
      PROCEDURE DIVISION.
          SAMPLE-PROGRAM.
     *    // Procedure PROC-01 does a complex database update.
     *    // If this fails then a restart should be mandatory.
     *    // To change the restart to "mandatory" (2), APD/400
     *    // API SETRST is used.

          MOVE "SETRST"    TO    APINAM  IN SETRST
          MOVE "00"        TO    APICCD  IN SETRST
          MOVE SPACES      TO    APIFTU  IN SETRST
          MOVE "2"         TO    RSTFL   IN SETRST

          CALL "QAFAPIPG"  USING SETRST IN QAFAPIPG
          PERFORM PROC-01

     *    // When procedure PROC-01 ends, the restart flag is set back
     *    // to "no-restart" (0) again using the APD/400 API SETRST.

          MOVE "0"        TO    RSTFL  IN SETRST
          CALL "QAFAPIPG" USING SETRST IN QAFAPIPG
```

SNDMSG Send Message

This API is used to send messages to APD/400 to be displayed on APD/400 displays. For example, a task performed from an APD/400 menu that does not require any user interaction sends a completion message. Both messages from message files and messages defined at program runtime can be sent.

To send a predefined message from a message file, the values for MSGFLIB, MSGF, and MSGID must be defined. MSGDTA contains the message data fields that replace the & variables defined for the message.

If field MSG contains any characters that are not blank, a message defined at program runtime is sent. If all fields contain only blanks, the completion code is 20.

To send a message defined at program runtime, the text to be displayed must be defined in field MSG.

Interface Description

Table 29. Parameters for the Send Message API (SNDMSG)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	SNDMSG The service name of the Send Message API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **20** The message file or the message file library does not exist. **21** The message does not exist in the specified message file.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Message file library MSGFLIB	*INPUT	CHAR(10)	10	17	The name of the library in which the message file is stored.
5 Message file MSGF	*INPUT	CHAR(10)	10	27	The name of the message file in which the message is stored.
6 Message ID MSGID	*INPUT	CHAR(7)	7	37	The identifier of the message in the message file.
7 Message data MSGDTA	INPUT	CHAR(512)	512	44	The data for the & variables in the message.
8 Message text MSG	INPUT	CHAR(132)	132	556	The text for messages defined at program run time. If field MSG is not blank, such a message is sent.
Total			**687**		

WRKDST Work with Data Sets

This API allows the developer or user to create, change, retrieve, and delete APD/400 data sets. It performs tasks similar to the `Administer Data Sets` function (see *IBM Application Program Driver/400 Version 3: Administrator's Guide*). For example, you can use the WRKDST API to:

- Provide multiple data sets during the post installation procedure
- Create identical data sets for an application family.

If an application library description (see "Adding an Application Library Description" on page 40) exists for the application, and the library name template contains at least one data set ID placeholder (`&Dn.`), the following apply:

- Application library descriptions with a data set placeholder in the library name template will be created or deleted if a data set is created or deleted. This is to guarantee that the symbolic names are resolved correctly for new data sets. However, no OS/400 object of type *LIB is created.
- A save/restore control record (see `Administering Control Records` in *IBM Application Program Driver/400 Version 3: Administrator's Guide*) is created or deleted whenever an application library description is created or deleted. The save/restore control record is created with the following values:

  Backup cycle: 1 (Daily)

  Starting date: Current date

  Priority: 00

  Generations: 3

  Use `Administer Control Records` to change these initial values.

"Sample Scenarios" on page 45 describes how this API can be used.

Interface Description

Table 30 (Page 1 of 3). Parameters for the Work with Data-Sets API (WRKDST)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	WRKDST The service name of the Work with Data-Sets API.

Table 30 (Page 2 of 3). Parameters for the Work with Data-Sets API (WRKDST)

Parameter	Use	Type	Len	Pos	Description
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **20** The specified installation does not exist. **21** The specified application does not exist. **22** You do not have authority for the application. You must be the administrator of the application (the APD/400 administrator can authorize you to use the Administer Applications function (ADMAPL)), or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **28** The operation code is not valid. **30** The specified data set does not exist. **42** The data-set description is blank. **43** The data-set name contains characters that are not valid. **44** The data set already exists. **45** The data-set description already exists. **46** The save object name is incorrect. **56** The authorization list does not exist. **99** Other errors. See the job log.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	*INPUT	CHAR(3)	3	17	The name of the installation. **Default:** The name of the installation to which the currently active task belongs.
5 Application name ANWID	*INPUT	CHAR(7)	7	20	The name of the application. **Default:** The name of the installation to which the currently active task belongs.
6 Data-set name FIRNR	*INPUT	CHAR(4)	4	27	The name of the data set. Valid characters are A-Z and 0-9. If the name of the data set is used in forming library names (replacement variables &Dn. are used in the library name template), characters used in the library name must not be blank. **Default:** The name of the currently active data set in the application to which the currently active task belongs.
7 Data-set description FIRNM	INPUT/ OUTPUT	CHAR(40)	40	31	The description of the data set. As an input parameter, it cannot be blank and it must be unique as a data-set description for this application in this installation. Input for operations 1 and 2, output for operation 3, not used for operation 4.

Table 30 (Page 3 of 3). Parameters for the Work with Data-Sets API (WRKDST)

Parameter	Use	Type	Len	Pos	Description
8 Authorization list BRFKT	INPUT/ OUTPUT	CHAR(10)	10	71	The authorization list controlling access to the data set. Blank means no authorization checking is performed. As an input parameter, the authorization list must have been previously defined. Input for operations 1 and 2, output for operation 3, not used for operation 4.
9 Operation code OPRCD	INPUT	CHAR(1)	1	81	The operation to be performed: **1** Create **2** Change **3** Retrieve **4** Delete.
Total			**81**		

Note: The setting Work with Data-Sets has no influence on API WRKDST.

WRKINS Work with Installations

This API allows the developer or user to create, change, retrieve, and delete APD/400 installations.

"Sample Scenarios" on page 45 describes how this API can be used.

Interface Description

Table 31 (Page 1 of 2). Parameters for the Work with Installations API (WRKINS)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	WRKINS The service name of the Work with Installations API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **20** The specified installation does not exist. **22** You do not have authority to work with installations. You must be an application administrator (the APD/400 administrator can authorize you to use the Administer Applications function (ADMAPL)), or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **28** The operation code is not valid. **37** The installation description is blank. **38** The installation name contains characters that are not valid. **39** The installation already exists. **40** The installation description already exists. **41** The installation cannot be deleted because applications still exist in it. **99** Other errors. See the job log.

Table 31 (Page 2 of 2). Parameters for the Work with Installations API (WRKINS)

Parameter	Use	Type	Len	Pos	Description
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Installation name INSID	*INPUT	CHAR(3)	3	17	The name of the installation. Valid characters are A-Z and 0-9. Embedded spaces are not allowed. **Default:** The name of the installation to which the currently active task belongs.
5 Installation description INSTX	INPUT/ OUTPUT	CHAR(40)	40	20	The description of the installation. As an input parameter, it cannot be blank and it must be unique as an installation description. Input for operations 1 and 2, output for operation 3, unused for operation 4.
6 Operation code OPRCD	INPUT	CHAR(1)	1	60	The operation to be performed: **1** Create **2** Change **3** Retrieve **4** Delete.
Total			**60**		

WRKSAVOBJ Work with Save Objects

This API allows the developer or user to create, change, retrieve, and delete APD/400 save object entries. It performs tasks similar to the Administer Control Records function (see *IBM Application Program Driver/400 Version 3: Administrator's Guide*).

If the save object type is 2 (Folder), only the save/restore control information is affected.

If the save object type is 1 (Library), the save/restore control information and the application library description (see “Adding an Application Library Description” on page 40) are affected as follows:

- If the operation code is 2 (Change), an application library description exists for the library, and blank has been specified for INSID, ANWID, and FIRNR, the application library description for the library is deleted.
- If the operation code is 2 (Change), an application library description exists for the library, and different values are specified for INSID, ANWID, and FIRNR, from those to which the application library description belongs, the application library description is moved to the specified installation, application, and data set.
- If the operation code is 4 (Delete), and an application library description exists for the library, the application library description is deleted.

All these operations can cause problems for an application that needs the application library description (for example, symbolic library names are resolved based on the information in the application library description). To avoid these problems,

retrieve (operation code 3) the save object attributes and test to see if an application library description exists for the library (in which case, INSID and ANWID are not blank). If an application library description exists, make sure that the application can run correctly without the application library description.

Interface Description

Table 32 (Page 1 of 3). Parameters for the Work with Save Objects API (WRKSAVOBJ)

Parameter	Use	Type	Len	Pos	Description
1 Service name APINAM	INPUT	CHAR(10)	10	1	WRKSAVOBJ The service name of the Work with Save Objects API.
2 Completion code APICCD	OUTPUT	CHAR(2)	2	11	**00** Successful completion. **01 - 19** Refer to "Completion Codes" on page 64 for more details on completion codes. **20** The specified installation does not exist. **21** The specified application does not exist. **22** You do not have authority to work with save object entries. You must be the application administrator (the APD/400 administrator can authorize you to use the Administer Applications function (ADMAPL)), or have the OS/400 special authorities *ALLOBJ and *SECADM (for example, by being the QSECOFR). **28** The operation code is not valid. **30** The specified data set does not exist. **46** The save object name is incorrect. **47** The save object type is incorrect. **48** The number of generations to be saved is incorrect. **49** The group priority is incorrect. **50** The medium type is incorrect. **51** The save cycle is incorrect. **52** The starting date for saving is incorrect. **53** The names of the installation, application, and data set must be blank when the save object type is not 1=Library. **54** The save object entry already exists. **55** The save object entry does not exist. **99** Other errors. See the job log.
3 Future use APIFTU			4	13	This field is reserved for future use. However, fill it with blanks (b = Hex 40) before you call the server.
4 Save object name BIBL01	INPUT	CHAR(12)	12	17	The name of the library or folder to be saved.

Table 32 (Page 2 of 3). Parameters for the Work with Save Objects API (WRKSAVOBJ)

Parameter	Use	Type	Len	Pos	Description
5 Save object type TYPE01	INPUT	CHAR(1)	1	29	The type of the object to be saved: **1** Library **2** Folder.
6 Generations to be saved GENE01	INPUT/ OUTPUT	CHAR(1)	1	30	Generations to be saved (1-9). Input for operations 1 and 2, output for operation 3, not used for operation 4.
7 Group priority PRIO01	INPUT/ OUTPUT	CHAR(2)	2	31	The priority for saving and restoring the save object. Input for operations 1 and 2, output for operation 3, not used for operation 4.
8 Medium type MEDT01	INPUT/ OUTPUT	CHAR(1)	1	33	The type of medium used to save the object: **1** Tape. Input for operations 1 and 2, output for operation 3, not used for operation 4.
9 Save cycle ZYKL01	INPUT/ OUTPUT	CHAR(1)	1	34	The save cycle for the save object: **0** Optional **1** Daily **2** Weekly **3** Monthly. Input for operations 1 and 2, output for operation 3, not used for operation 4.
10 Starting date for saving STDT01	*INPUT/ OUTPUT	CHAR(8)	8	35	The starting date to save the object (YYYYMMDD). **Default:** Today's date. Input for operations 1 and 2, output for operation 3, not used for operation 4.
11 Installation name INSID	*INPUT/ OUTPUT	CHAR(3)	3	43	The name of the installation. Blank if the save object does not belong to a particular application under control of APD/400. **Default:** The name of the installation to which the currently active task belongs. Input for operations 1 and 2, output for operation 3, not used for operation 4.
12 Application name ANWID	*INPUT/ OUTPUT	CHAR(7)	7	46	The name of the application. Blank if the save object does not belong to a particular application under control of APD/400. **Default:** The name of the installation to which the currently active task belongs. Input for operations 1 and 2, output for operation 3, not used for operation 4.

Table 32 (Page 3 of 3). Parameters for the Work with Save Objects API (WRKSAVOBJ)

Parameter	Use	Type	Len	Pos	Description
13 Data-set name FIRNR	*INPUT/ OUTPUT	CHAR(4)	4	53	The name of the data set. Blank if the save object does not belong to a particular application under control of APD/400, if the application does not use data sets, or if the save object does not belong to a data set of the application. **Default:** The name of the currently active data set in the application to which the currently active task belongs. Input for operations 1 and 2, output for operation 3, not used for operation 4.
14 Operation code OPRCD	INPUT	CHAR(1)	1	57	The operation to be performed: **1** Create **2** Change **3** Retrieve **4** Delete.
Total			**57**		

Appendix A. Layout of File QAAFTASK0

Each record in the task file QAAFTASK0 holds an entry for a task. A task is an element function within APD/400. A record in QAAFTASK0 holds all data that APD/400 needs to process the task.

Note: This interface is as described in “Interfaces for Version 3 Release 6” on page viii.

Record Layout

The following table shows the record layout of the QAAFTASK0 file:

Table 33 (Page 1 of 2). QAAFTASK0: Task File

#	Field Name	Data Type	Bytes	Offset	Key	Default	Description
1	INSID	CHAR(3)	3	1	1A	*none	Installation ID
2	ANWID	CHAR(7)	7	4	2A	*none	Application ID
3	MTASK	CHAR(10)	10	11	3A	*none	Task ID/Expert code
4	TASKA	CHAR(1)	1	21		*none	Task type
5	EXTYP	CHAR(1)	1	22		*none	Processing type
6	HLDOC	CHAR(12)	12	23		*spaces	Help document
7	HLLAB	CHAR(10)	10	35		*spaces	Help label
8	BRFKT	CHAR(10)	10	45		*spaces	Authorization list ID
9	PGMGR	CHAR(10)	10	55		*spaces	Exclusion list ID
10	MAWTX	CHAR(46)	46	65		*none	Menu option text
11	AUDTF	CHAR(1)	1	111		*spaces	Audit flag
12	MAURR	CHAR(1)	1	112		*spaces	Control flag
13	MPPRT	CHAR(512)	512	113		*spaces	Menu program parameter
14	MPGMN	CHAR(10)	10	625		*none	Menu program name
15	RSTFL	CHAR(1)	1	635		*none	Restart flag
16	ENVID	CHAR(10)	10	636		*spaces	Environment ID
17	OVRSD	CHAR(1)	1	646		0	Overwrite schedule date
18	OVRBE	CHAR(1)	1	647		0	Overwrite batch environment
19	OVRPR	CHAR(1)	1	648		0	Overwrite parameter
20	PRPGM	CHAR(10)	10	649		*spaces	Parameter definition user-exit program
21	CNFMS	CHAR(1)	1	659		0	Send confirmation message
22	SCDMS	CHAR(1)	1	660		0	Send schedule message
23	ISCHT	CHAR(3)	3	661		Hex hhmmss	Initial schedule time
24	ISTAT	CHAR(3)	3	664		RDY	Initial schedule status
25	MHFID	CHAR(10)	10	667		*spaces	Menu headings format ID
26	ULPRM	CHAR(2)	2	677		*spaces	Interface level for the user-exit program for parameter definition (PRPGM)

Table 33 (Page 2 of 2). QAAFTASK0: Task File

#	Field Name	Data Type	Bytes	Offset	Key	Default	Description
27	MSTXT	CHAR(20)	20	679		*spaces	Task short text
28	ACTBR	CHAR(10)	10	699		*spaces	Related menu bar
29	MNUWD	CHAR(1)	1	709		1	Menu window
	Total		**709**				

Extended Field Descriptions

The following are detailed descriptions of the fields in the QAAFTASK0 file:

INSID The 3-digit installation ID.

ANWID The 7-digit application ID is the program number of the application (for example, *APD for APD/400).

MTASK The expert code name that identifies a task within an application.

TASKA The task type can be:

- **B** A menu bar task.
- **C** A command task. For command tasks, APD/400 interprets the task name `MTASK` as an OS/400 command with the associated parameters stored in the menu parameter `MPPRT`. A command is *not* processed through the AIP. This prevents a user from being granted all administrator authorizations for the corresponding application, because the AIP is always compiled with the USRPRF parameter of the CRTxxx command set to *OWNER.
- **D** A pull-down task.
- **L** A process list task.
- **M** A menu task.
- **P** A program task.

EXTYP The processing type can be `I` for interactive or `B` for batch. `EXTYP=B` is not allowed for pull-down, menu bar, and menu tasks.

HLDOC If the Help source for the task is:

- OfficeVision/400, this field is the name of a document.
- UIM, this field is the name of a panel group.
- An OS/400 library, this field is the name of a display file.
- A user-exit program, this field is the name of an identifier.

HLLAB If the Help source for the task is:

- OfficeVision/400, this field is the name of a label.
- UIM, this field is the name of a Help module.
- An OS/400 library, this field is the name of a record.
- A user-exit program, this field is the name of an identifier.

BRFKT The name of an authorization list. Only users that have an authorization of at least 1 in this list are authorized to perform the task.

PGMGR A task of type program (`TASKA=P`) can be associated with a program group. A program group is stored in the program group text file (QAAFPGTX0) and the program group file (QAAFPRGR0). A program group is identified with `INSID/ANWID/PGMGR`.

MAWTX The menu option text is a brief description of the task.

AUDTF If this field contains `0`, no audit records are written when the task is processed. If this field contains `1`, records are written into the Audit file. If this field is empty, the related value from the Application file is used.

MAURR This control flag has no specific use for APD/400 itself. It is passed unchanged to the AIP, where it can be used in several ways. The value for this field can be `1` for Yes or `0` for No.

MPPRT The menu program parameter holds all the information that an application needs to run a program. The 512 bytes can be defined as needed by the application. This field can be changed, depending on the value of the field `OVRPR`.

MPGMN The name of the program if the task is type P (program).

RSTFL The restart flag defines the way in which APD/400 controls the processing of a task. This flag can have the following values:

0 No restart control. The user is not notified if the task ended abnormally.

1 Normal restart control. APD/400 notifies the user if a task with this definition fails. The restart is displayed but can be omitted on `Work with Canceled Jobs`.

2 Mandatory restart. A task defined as mandatory restart cannot be omitted within `Work with Canceled Jobs`. It may be set to HLD if the user wants to resolve the restart situation later.

ENVID The name of a batch environment.

OVRSD This field holds a Boolean value that defines whether or not the schedule date can be overwritten during the invocation and the changing of a batch job:

0 The schedule date cannot be overwritten.
1 The date can be overwritten.

OVRBE This field holds a Boolean value that defines whether or not the batch environment can be overwritten during the invocation and the modification (using `Work with Scheduled Jobs`) of a batch job:

0 The batch environment cannot be overwritten.
1 The batch environment can be overwritten.

APD/400 checks whether the user is authorized to use the specified batch environment at invocation and modification time.

OVRPR This field holds a Boolean value that defines whether or not the task parameter can be overwritten during the invocation and the changing of a batch job:

0 The batch task parameter cannot be overwritten.
1 The batch task parameter can be overwritten.

PRPGM If this value is defined, the editing of the task parameter (`MPPRT`) in the batch scheduler and `Work with Scheduled Jobs` functions is done using a special program. This program is part of the application and is called via the AIP of the application.

CNFMS This is a 1-byte Boolean field and can have two values:

0 No confirmation message is sent.

1 A confirmation message is sent when the user wants to invoke a batch job via APD/400 and no further panels are displayed. If the user presses F3 or F12, the batch job is not submitted. If the user presses Enter, the batch job is submitted to the APD/400 batch handler.

SCDMS This is a 1-byte Boolean field and can have two values:

0 No message is displayed.

1 APD/400 sends a message to the user stating that the batch task has been transferred to the APD/400 batch handler.

ISCHT The initial schedule time. The value in this field is used as the initial value for the field `SCHTM` in QAAFSCDL0. If field `OVRSD` is set to `1`, the suggested schedule time can be overwritten by the user at schedule time, and by using the function `Work with Scheduled Jobs`. The format of this field is `hhmmss`.

ISTAT The initial schedule status. A record with a value in this field as the initial state is inserted into the schedule table when the task is scheduled. The value of this field can be:

RDY The task is ready to be submitted. At the appropriate date and time, APD/400 processes the task.

HLD The task will not be submitted until the state is changed to `RDY` using the `Work with Scheduled Jobs` function. Initialize your batch jobs with `HLD` if you do not want them to run immediately; for example, if you want report jobs to run only during the night but you want to allow the scheduling during the day.

MHFID This field points to an entry in the menu headings format file (QAAFMHFS0). It is applicable only for menu tasks (`TASKA=M`). If this field is left blank, the default heading for the application (stored in field `MHFID` in QAAFANWG0) to which the task belongs is used. If that field is blank, the APD/400 default menu heading is used.

ULPRM This field defines the layout level of the parameters passed by APD/400 to the user exit specified in `PRPGM`. Valid values are `00` through `99`. If this is not specified, APD/400 uses the layout for the current release level.

MSTXT A short text of up to 20 characters describing the task.

ACTBR This field is used only for menu tasks (`TASKA=M`). It specifies the name of the menu bar to be used for the menu. The menu bar must be in the same application (same `ANWID`) as the menu.

MNUWD This field indicates whether the menu is displayed in a window or full-screen. The user's initial menu is always displayed as a full-screen menu, regardless of the value in this field.

Appendix B. Layout of File QAAFMENU0

The menu file QAAFMENU0 defines the structure of menus, pull-downs, and process lists. It contains one record for each menu option, pull-down choice, or list entry. Each menu, pull-down, or process list and each option, choice, and entry points to a record in the task file (see Appendix A, "Layout of File QAAFTASK0" on page 101).

Notes:

1. If ANWID or ANWMT are equal to *APD, the corresponding entry in the task file must be searched with the installation ID (INSID) of ' ', regardless of the settings of INSID in QAAFMENU0.
2. This interface is as described in "Interfaces for Version 3 Release 6" on page viii.

Record Layout

The following table shows the record layout of the QAAFMENU0 file:

Table 34 (Page 1 of 2). QAAFMENU0: Menu File

#	Field Name	Data Type	Bytes	Offset	Key	Default	Description
1	INSID	CHAR(3)	3	1	1A	*none	Installation ID.
2	ANWID	CHAR(7)	7	4	2A	*none	Application ID of the menu.
3	MENAM	CHAR(10)	10	11	3A	*none	Menu name. The task ID of the menu. The description and other information about the menu must be stored in the Task file.
4	MAWNR	PACKED(2)	2	21	4A	*none	Menu selection number. A maximum of 14 menu selections can be defined in one menu. The valid range is 01 through 99.
5	ANWMT	CHAR(7)	7	23		*none	Application ID of the task. The internal application ID of the selected task. ANWID and ANWMT are identical if the menu and the menu selection belong to the same application.
6	MTASK	CHAR(10)	10	30		*none	Task name/expert code. The name (expert code) of the selected task. It can be of any task or processing type except that, for process lists, only the last item in the list can be a menu task.

Table 34 (Page 2 of 2). QAAFMENU0: Menu File

#	Field Name	Data Type	Bytes	Offset	Key	Default	Description
7	MSHTP	CHAR(1)	1	40	5A	*none	Menu subheading type. Describes whether this entry is used as a menu option that points to a task in the Task file (MSHTP=1), or is used to store subheading information (MSHTP=0), or is used as an option on a menu bar (MSHTP=2). If the task identified by field MENAM is B (menu bar), D (pull-down), or L (process list), the value in this field must be 1.
8	MSHTX	CHAR(46)	46	41		*spaces	Menu subheading text. Contains the text for the subheading if MSHTP=0. It can contain a blank line. If MHSTP=2, this field contains the menu bar choice text. In this case, it must not be blank.
	Total		**86**				

On the installation tape, this file (renamed to QAAFMENUA) must contain an additional record that APD/400 uses to insert your application into the APD/400 main menu at the first free position. Leave fields INSID and MENAM blank, enter 0 for field MAWNR, *APD for field ANWID, and the internal ID of your application in field ANWMT. MTASK is the name of your main menu. Without an entry, the application is not listed in the APD/400 main menu, but can be invoked using the expert code.

Appendix C. Adding Tasks to the Task File

This is an example of a program that inserts records into the APD/400 task file (QAAFTASK0). Each record in the file represents a task. The tasks created with this program allow you to use OS/400 commands from the APD/400 command line.

Note: This interface is as described in "Interfaces for Version 3 Release 6" on page viii.

To run this program, you must have SQL/400 on your system, and you need *ALLOBJ authority:

1. Use the `Administer Applications (Developer)` function (ADMAPP) to define an application in APD/400. The external name of the application could be, for example, OS, and the internal name 5738SS1. You can use one of the sample AIPs provided (see "Sample AIP" on page 20, and "Additional Sample AIPs" on page 21) as the basis on which to build an AIP for your application.
2. In OS/400, type in the following command to create a file DSPOBJD in QTEMP with the list of all OS/400 commands:

```
DSPOBJD OBJ(QSYS/*ALL) OBJTYPE(*CMD *MENU)
  OUTPUT(*OUTFILE) OUTFILE(QTEMP/DSPOBJD)
```

3. Start SQL/400 by typing:

```
STRSQL
```

4. Type in the following SQL/400 statements:

```
INSERT INTO QUSRSYS/QAAFTASK0
  (ANWID, MTASK, TASKA, EXTYP, MAWTX, MAURR, MPGMN,
      RSTFL)
  SELECT '5738SS1', ODOBNM, 'P', 'I',
      SUBSTR(ODOBTX, 1, 46), '1', ODOBNM, '0'
    FROM QTEMP/DSPOBJD
    WHERE ODOBTP = '*CMD'
      AND ODOBNM <> 'DATA'

INSERT INTO QUSRSYS/QAAFTASK0
  (ANWID, MTASK, TASKA, EXTYP, MAWTX)
  SELECT '5738SS1', ODOBNM, 'M', 'I',
      SUBSTR(ODOBTX, 1, 46)
    FROM QTEMP/DSPOBJD
    WHERE ODOBTP = '*MENU'
      AND ODOBNM NOT LIKE 'CMD%'
INSERT INTO QUSRSYS/QAAFTASK0
  (ANWID, MTASK, TASKA, EXTYP, MAWTX, MAURR, MPGMN,
    MPPRT, RSTFL)
  SELECT '5738SS1', ODOBNM, 'P', 'I',
      SUBSTR(ODOBTX, 1, 46), '0', 'GO', ODOBNM, '0'
    FROM QTEMP/DSPOBJD
    WHERE ODOBTP = '*MENU'
      AND ODOBNM LIKE 'CMD%'
```

All system commands and menus are now directly accessible as expert codes from any APD/400 menu (for example, OS/DSPMSG).

If you want the prompting for a particular command, use the `Administer Menus` function (ADMMNU) to change the task that represents that command.

Where necessary, you can improve the conversion of OS/400 to APD/400 by analyzing the output of the DSPCMD command (for example, by inserting only commands that can be processed by QCMDEXC in an interactive production environment).

Note: No menus for OS/400 commands have been created. You cannot read the system tables in which information about OS/400 menus is stored. You can, however, manually call each system menu, press F1 (Help) and then press F14. This creates a spool file that you can analyze in a program (after you have run the CRTSPLF command) to create the menu selections.

You can now use the `Administer Menus` function to build menus using the tasks you have created.

Appendix D. Evaluating the APD/400 Audit File

This appendix contains general-use programming interface and associated guidance information.

The APD/400 Audit file (QAAFAUDT0) contains a record for each event in APD/400. An event is, for example, the starting or ending of a task. This file is written sequentially and is not used by APD/400 for retrieving information.

The following table shows the layout of the Audit file:

Table 35. QAAFAUDT0: Audit File

Field Name	Length	Type	Description
USRID	10	A	User ID
JOBID	10	A	Job ID
JOBNR	6	A	Job number
EVTDS	4	A	Event date (SQL) internal
EVTTS	3	A	Event time (SQL) internal
EVTDA	8	A	Event date alpha
EVTTA	6	A	Event time alpha
EVTDD	10	A	Event date external
EVTTD	8	A	Event time external
AKTST	3,0	P	Activity level
UNTST	3,0	P	Subprogram level
INSID	3	A	Installation ID
ANWID	7	A	Application ID
MTASK	10	A	Task ID/Expert code
FIRNR	4	A	Data-set ID
RETCD	1	A	Return code
EVNTP	7	A	Event type
BJSNR	9,0	P	Batch job sequence number
RSSNR	9,0	P	Restart sequence number
EXTYP	1	A	Processing type

The fields `USRID`, `JOBID`, and `JOBNR` make up the unique identifier for a job in OS/400.

USRID The name of the user who owns the job. For interactive jobs, this is the name of the user who signed on to the system. For batch jobs, this is the name of the user who invoked the job or the user specified in the `ENUSR` field of the related batch environment.

JOBID The name of the job. For interactive jobs, this is the name of the work station where the job was invoked. For batch jobs, APD/400 uses the name of the `MTASK` field in the Task file (QAAFTASK0).

JOBNR The job number is a 6-digit number assigned sequentially by OS/400 to prevent duplicate job IDs.

EVTDS The event date in APD/400 internal representation (SQL/400 format).

EVTTS The event time in APD/400 internal representation (SQL/400 format).

EVTDA The event date in the format `YYYYMMDD`. This field is a character field with one byte per digit. It can be used as a sort or selection field for AS/400 Query.

EVTTA The event time in the format `HHMMSS`. This field is a character field with one byte per digit. It can be used as a sort or selection field for AS/400 Query.

EVTDD The event date format for displays. This format depends on the value of the APD/400 parameter APD_DATE_REPRESENTATION.

EVTTD The event time format for displays.

AKTST The activity level is used as a key for the link between the Jobs file (QAAFJOBS0) and the Main Program file (QAAFANAS0). The currently highest activity level is stored in the User file QAAFANWR0 (field `AKTST`), and is updated by the online monitor program QAFDRMAIN. This field is a constant filler that is set to `0`.

UNTST The subprogram level information shown in this field is used to identify the number of active APD/400 calls within this job, and is used for all related files (Subprogram file QAAFANUS0 and Stack file QAAFSTCK0) to uniquely identify a record. This field is a constant filler that is set to `0`.

INSID The 3-digit installation ID.

ANWID This field is usually the application ID that uniquely defines an application within an installation ID. For IBM applications, the IBM program number (without the dash) is the application ID.

MTASK The task ID (expert code) identifies a task within an application. This name is used as a reference for several purposes; as an expert code, for building menus, and so on.

FIRNR The data-set ID is recorded if the application to which the task belongs is designed to work with different data-set IDs. If not, this field is left empty.

RETCD A return code is stored for entries with `EVNTP=END`, `CNL`, or `RST`. This is the return code that the online monitor program QAFDRMAIN or the Batch Gate Program (BGP) receives from the AIP.

EVNTP The event type can have the following values:

- **ACT** Marks a start entry.
- **END** Marks an end entry.
- **RST** Marks a restart entry, which is a task that has been defined as capable of restart (field `RSTFL=1` or `2` in file QAAFTASK0), and ends with a return code greater than 2 (abnormal end or cancelation).
- **CNL** If a task not defined as restartable ends with a return code of greater than 0, 1, or 2, the system writes a CNL entry to the Audit file.

RST-CNL A task canceled with option 4 (Delete) on `Work with Canceled Jobs` causes a RST-CNL entry.

RST-ACT A task marked as a restart and restarted using `Work with Canceled Jobs` causes a RST-ACT entry.

AUT If you attempt to invoke a task for which you are not authorized, the system adds this entry to the Audit file.

EXC An interactive task excluded by another user, or a batch task marked as excluded after several rescheduling attempts, causes an EXC entry.

RSC A batch task that has been excluded is rescheduled several times. The number of rescheduling attempts and the length of the intervals between them can be set in the Parameter file. Tasks that are being rescheduled cause RSC entries.

BJSNR This job sequence number is used to uniquely identify an invocation of a batch or interactive job via APD/400.

RSSNR This is the batch job sequence number for those jobs for which the current job is a restart. It indicates the first job in a possible chain of started, failed, and restarted jobs. This makes it possible to retrieve the entries in the Audit file in that chain by grouping the entries according to their `RSSNR` value.

EXTYP The processing type can be `I` (interactive) or `B` (batch).

Creating Audit File Query Reports

In the Audit file (QAAFAUDT0), APD/400 stores information about how often specific tasks are called. This requires that the user specified `Y` as the audit flag for the tasks using `Administer Menus`. If you do not specify `1` as the value for the audit flag in the Task file, the default value for task auditing, specified in the Application file, is used.

Auditing provides you with statistical material about system activities. Even if you are not familiar with AS/400 Query, the two examples that follow will help you in using it to produce the corresponding report and to create similar queries. IBM AS/400 Query must be installed on your system if you want to change the supplied queries. For more information, see the *AS/400 Query: User's Guide*.

Example 1

You want to know which administration functions have been called during a specific period of time (1993-02-02 until 1993-02-21), and how many calls occurred for the different companies of the default installation ‘ ’. The expert code of any administration function starts with ADM.

Note: You use different data sets in your application to separate data for different companies you have as clients.

This sample query is stored as QAF_QD01 in library QUSRSYS. It queries file QAAFAUDT0 in library *LIBL. The report may look as follows. The data-set ID is interpreted in this example as the ID of a company.

```
93/02/22  09:59:44  PAGE    1

      Task         Company ID
      name/expert
      code

      ADMAPL

Menu ADMAPL in company   called:
COUNT 7            7

Menu ADMMNU in company   called:
COUNT 2            2

In company   menus were called:
COUNT 9            9

      ADMAPL       0001

Menu ADMMNU in company 0001 called:
COUNT 4            4

In company 0001 menus were called:
COUNT 4            4

      ADMAPL       1234

Menu ADMAPL in company 1234 called:
COUNT 1            1

In company 1234 menus were called:
COUNT 1            1

Menus called in all companies
COUNT 14           14

* * * E N D  O F  R E P O R T * * *
```

You could produce a similar report by using the following SQL/400 command:

```
SELECT MTASK, FIRNR FROM QAAFAUDT0 WHERE INSID =
'  'AND EVTDA BETWEEN '19930201' AND'19930221' AND
MTASK LIKE 'ADM%%%' ORDER BY FIRNR, MTASK
```

Example 2

This example is stored as QAF_QD02 in library QUSRSYS. The report produced by this query tells you how many menus have been called interactively or in batch mode on a specific day (1993-01-18).

The Query report may look as follows:

```
93/02/22  16:27:58            PAGE 1

Event date  Processing type Task
                            name/expert
                            code

1993-01-18    B             EXT01
1993-01-18                  EXT01
1993-01-18                  SCHED001
1993-01-18                  SCHED001
1993-01-18                  TESTPROGB
1993-01-18                  TESTPROGB
1993-01-18                  TESTPROGB
1993-01-18                  TESTPROGB
1993-01-18                  TESTPROGB
1993-01-18                  TESTPROGB

       Processing type B used:
       COUNT 10

1993-01-18   I              TESTCMDI
1993-01-18                  TESTCMDI
1993-01-18                  TESTPROGI
1993-01-18                  TESTPROGI
1993-01-18                  TESTPROGI
1993-01-18                  WRKCNLJOB
1993-01-18                  WRKSBMJOB
1993-01-18                  WRKSBMJOB
1993-01-18                  WRKSBMJOB
1993-01-18                  WRKSCDJOB

       Processing type I used:
       COUNT 10

       Total number of calls on specified day:
       COUNT 20

* * * E N D  O F  R E P O R T * * *
```

You could produce a similar report by using the SQL/400 command:

```
SELECT EVTDA, EXTYP, MTASK FROM QAAFAUDT0 WHERE
EVTDA = '19930118' ORDER BY EXTYP, MTASK
```

Glossary of Terms and Abbreviations

This glossary defines terms used in the APD/400 library.

A

AIP. Application interface program.

API. Application program interface.

application. A program used to perform a particular data processing task, such as inventory control or payroll.

application interface program (AIP). A functional interface used by APD/400 to invoke the corresponding application.

application program interface (API). A functional interface that allows an application program written in a high- or low-level language to use specific data or functions of APD/400.

authorization. The process of giving a user either complete or restricted access to an object, resource, or function.

authorization list. Authorization lists are used to protect menus, menu options, installations, and data sets from unauthorized access. An authorization list consists of a 10-digit authorization list name and a list of authorization list entries, each of which is comprised of a user name and an authority level.

B

back up. To save some or all of the objects on a system to tape or diskette, for safe keeping.

backup. (1) Pertaining to an alternative copy used as a substitute if the original is lost or destroyed, such as a backup log. (2) The act of saving some or all of the objects on a system to a tape, diskette, or save file. (3) The tapes, diskettes, or save files with the saved objects.

Batch Monitor Program (BMP). The program used by APD/400 to control batch jobs.

batch processing. A method of running a program or a series of programs in which one or more records (a batch) are processed with little or no action from the user or operator. Contrast with *interactive processing*.

BMP. Batch Monitor Program.

C

CL. Control language.

command. A statement used to request a function of the system. A command consists of the command name abbreviation, which identifies the requested function, and its parameters.

Common User Access (CUA). Pertaining to a Systems Application Architecture (SAA) specification that gives a series of guidelines describing the way information should be displayed on a screen, and the interaction techniques between users and computers.

control language. The set of all commands with which a user requests system functions.

control record. The description of a save operation for libraries and folders. Record details include the save cycle, the number of generations to be stored, the medium, and the starting date.

CUA. Common User Access.

D

data description specifications (DDS). A description of the user's database or device files that is entered into the system in a fixed form. The description is then used to create files.

data set. A data environment contained within an application. For applications that are multi-data set enabled, data sets could be used for different departments or clients.

DBCS. Double-byte character set.

DDS. Data description specifications.

default. A value that is automatically supplied or assumed by the system or program when no value is specified by the user.

DLO. Document library object.

document library object (DLO). Any system object that resides in the document library, such as RFT and FFT documents, folders, and PC files.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets.

Because each character requires 2 bytes, the typing, displaying, and printing of DBCS characters requires hardware and programs that support DBCS.

E

exclusion. An exclusion defines which programs cannot be active simultaneously in an application.

exclusion list. An exclusion list combines tasks into groups. Defining exclusion lists saves the work of specifying each program for exclusions. Programs with identical exclusion characteristics can be combined in exclusion lists. An exclusion list can be either a function list or an object list.

expert code. A command or abbreviation used to invoke a menu or program. Entering an expert code allows the user to directly access a menu, task, or program (without calling intermediate menus), as well as to switch between applications without signing off and on. Pressing F4 on a menu screen displays a list of all expert codes for which the user is authorized.

F

folder. A directory for documents. A folder is used to group related documents and to find documents by name. The system-recognized identifier for the object type is *FLR. Compare with *library*.

function key. A keyboard key that allows the user to select keyboard functions or programmer functions. The keys available are displayed on line 23 of the screen display.

H

help function. Pressing either the Help key or F1 on a display provides information on a specific part of that display or the whole display, depending on the position of the cursor. If the cursor is located on a message, second-level text for that message is displayed.

high-level language (HLL). A programming language, such as RPG, BASIC, PL/1, Pascal, COBOL, and C used to write computer programs.

HLL. High-level language.

I

IBM SAA OfficeVision/400 Version 3. The IBM licensed program that allows users to prepare, send, and receive mail; schedule items on calendars; maintain directories of names and addresses; file and retrieve documents; and create and maintain distribution lists. SAA OfficeVision/400 also provides word processing functions and the capability to work on behalf of other users.

initial menu. The first menu displayed after sign-on. Each user can set a personal initial menu. Pressing F23 on a menu labels it as the initial menu for the user.

installation. A specific processing environment containing applications and data sets. For example, different installations could be used for test and production environments.

interactive processing. A processing method in which each operator action causes a response from the program or the system. Contrast with *batch processing*.

J

job. (1) A unit of work to be done by a computer. (2) In the SAA OfficeVision/400 calendar function, an item that schedules a control language (CL) command to run at any date or time.

journal. A system object that identifies the objects being journaled, the current journal receiver, and all the journal receivers on the system for the journal. The system-recognized identifier for the object is *JRN.

journal receiver. A system object that contains journal entries added when changes are made to an object, for example, when an update is made to a file being journaled. The object type is *JRNRCV.

journaling. The process of recording, in a journal, the changes made to a physical file member.

L

language priority list. In APD/400, a list of available languages ranked in order of a user's preference, to enable the use of an application in one of the preferred languages. Number 1 on the priority list is the user's most preferred language.

library. A system object that serves as a directory to other objects. A library groups related objects, and allows the user to find objects by name. The system-recognized identifier for the object type is *LIB. Compare with *folder*.

library list. A list that indicates which libraries are to be searched and the order in which they are to be searched. The system-recognized identifier is *LIBL.

library name template. Naming conventions used in combination with the installation and data-set IDs to resolve a library name during installation of an applica-

tion that supports multiple installations or multiple data sets.

local data area. A 1024-byte data area that can be used to pass information between programs in a job. A separate local data area is automatically created for each job.

M

menu. A displayed list of items from which a user can make a selection. The system-recognized identifier for the object type is *MENU.

menu bar. The area containing keywords at the top of a display that gives a user access to actions available for that display. After a user requests a choice in the menu bar, a pull-down menu is shown below the menu bar.

message. A communication sent by the AS/400 system or APD/400, and displayed either on line 24 of the screen or in window format.

MNCS. Multinational character set.

More. When displays and menus comprise several pages, `More...` is displayed at the bottom of all screens except the last one.

Position your cursor on the display, or message line, and press F8. You can continue to do this until the word `Bottom` is displayed.

multilingual support. Support that includes more than one national language on a system. See also *National Language Support*.

multinational character set (MNCS). A set of graphic characters that support the languages within a specific language group.

N

National Language Support (NLS). The ability for a user to communicate with hardware and software products in a language of choice to obtain results that are culturally acceptable. See also *multilingual support*.

NLS. National Language Support.

O

object. A named storage space that consists of a set of characteristics that describe itself and, in some cases, data. An object is anything that exists in and occupies space in storage and on which operations can be performed. Some examples of objects are programs, files, libraries, and folders.

OfficeVision. See *IBM SAA OfficeVision/400 Version 3*.

Operational Assistant. Pertaining to a part of the operating system that provides a set of menus and displays for end users to do commonly performed tasks, such as working with printer output, messages, and batch jobs.

output queue. An object that contains a list of spooled files to be written to an output device, such as a printer or a diskette. The system-recognized identifier for the object type is *OUTQ.

P

parameter. A value supplied to a command or program that is used either as input or to control the actions of the command or program.

process list. A list of program or command tasks that are automatically processed in sequence. The last entry can be a menu. A process list is a special type of menu in which all tasks are called. Only when the last task has been processed is control returned to the previous menu or the menu at the end of the process list.

program. A sequence of instructions that a computer can interpret and run.

pull-down menu. An extension of the menu bar that displays a list of available choices for a choice selected by a user in the menu bar. After a user selects a choice in the menu bar, the pull-down menu is shown.

Q

queue. A list of messages, jobs, files, or requests waiting to be read.

R

restart. The action necessary when a batch job has failed to run as scheduled.

restore. To copy data from tape, diskette, or a save file to auxiliary storage. Contrast with *save*.

S

SAA. Systems Application Architecture.

save. To copy specific objects, libraries, or data by transferring them from main storage or auxiliary storage to a media such as tape, diskette, or a save file. Contrast with *restore*.

scheduled job. A batch job that becomes eligible to run at a specified date and time.

scroll. To move a display image vertically or horizontally to view data that cannot be seen within the boundaries of the displayed screen.

SEU. Source entry utility.

source entry utility (SEU). A function of the AS/400 Application Development Tools licensed program that is used to create and change source members.

spooled file. A file that holds output data waiting to be processed, such as information waiting to be printed.

start program. A program defined by a user to run at sign-on time.

symbolic library name. A name given to represent a library in an application that supports multiple installations or multiple data sets. A symbolic library name is used in conjunction with a library name template.

Systems Application Architecture (SAA). Pertaining to an architecture defining a set of rules for designing a common user interface, programming interface, application programs, and communications support for strategic operating systems such as the OS/2, OS/400, VM/370, and MVS/370 operating systems.

T

tape cartridge. A case containing a reel of magnetic tape that can be put into a tape unit without stringing the tape between reels.

tape drive. A device used to move the tape and read and write information on magnetic tapes.

task. A basic unit of work to be defined. In APD/400 there are six task types:

- Command
- Menu
- Menu bar
- Process list
- Program
- Pull-down.

textual data. The collective term for menus, displays, lists, prompts, options, online Help information, and messages.

timetable. A schedule showing a planned order or sequence, used in APD/400 to determine when to run a recurring batch job.

toggle. Pertaining to a switching device, such as a toggle key on a keyboard, that allows a user to switch between two types of operations. For example, in APD/400 you can press F11 to display expert codes, and press the key again to return to the original screen display.

U

UIM. User Interface Manager.

user exit. A program routine given control by APD/400 to enhance services provided by APD/400 functions.

user identification (user ID). The name used to associate the user profile with a user when the user signs on the system.

User Interface Manager (UIM). A function of the operating system that provides a consistent user interface by providing comprehensive support for defining and running panels (displays), dialogs, and online Help information.

user password. A unique string of characters that a system user enters to identify that user to the system, if the system resources are secured.

user profile. An object with a unique name that contains the user's password, the list of special authorities assigned to the user, and the objects the user owns. The system-recognized identifier for the object type is *USRPRF.

V

value. Data (numbers or character strings) entered in an entry field, and data supplied in parameters of CL commands.

W

window. A part of the display screen with visible boundaries in which information is displayed. For example, in APD/400 when F17 (Position to) is pressed (where applicable), the `Position the List` window is displayed over the current display.

Bibliography

This bibliography lists related APD/400 publications and other documentation that provides general information.

IBM Application Program Driver/400 Version 3 publications

- *Administrator's Guide*, SH12-6403-00
- *General Information*, GH12-6401-00
- *Licensed Program Specifications*, GH12-6400-00
- *User's Guide*, SH12-6402-00.

Other publications

- *AS/400 Guide to Programming Application and Help Displays*, SC41-0011
- *AS/400 National Language Support* , SC41-4101
- *AS/400 Control Language Programming*, SC41-4721
- *AS/400 Query: User's Guide*, SC41-4210.

Index

D

E

F

H

I

J

L

M

O

P

Q

R

S

T

U

V

W

Your comments, please ...

IBM Application Program Driver/400 Version 3
Developer's Guide
Release 6.0

Publication No. SH12-6404-00

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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

If you mail this form to us, be sure to print your name and address below if you would like a reply.

You can also send us your comments using:

- A FAX machine. The number is: +49–511–5165340.
- Internet. The address is: gadlid@sdfvm1.vnet.ibm.com.
- IBMLink. The address is: SDFVM1(GADLID).
- IBM Mail Exchange. The address is: DEIBM3P3 at IBMMAIL.

  Please include the title and publication number (as shown above) in your reply.

Name _____ Address _____

Company or Organization _____ _____

Phone No. _____ _____

Your comments, please ...
SH12-6404-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Anwendungssysteme GmbH
Information Development, Dept. 5160
Postfach 72 12 80
30532 Hannover
Germany

Fold and Tape **Please do not staple** Fold and Tape

SH12-6404-00

Cut or Fold
Along Line

File Number: AS400-79
Program Number: 5716-PD1

Printed in Denmark

SH12-6404-00