

Planning a PAC Environment

Predict Application Control (PAC) is a system for controlling and tracking changes made to Natural or "foreign" (non-Natural and non-Predict) applications throughout the application life-cycle. In the production environment, PAC uses the Predict Application Audit (PAA) subsystem to protect and audit the application.

This chapter covers the following topics:

- Application Phases
- Integrating PAC
- The PAC Controlled Environment
- When to Implement PAC
- What PAC Functions to Implement
- Life-Cycle of a PAC Object
- Foundations of PAC Control
- PAC Entities and Their Relationships
- PAC Administration Facilities
- PAC Utilities
- Facilities for Customizing PAC

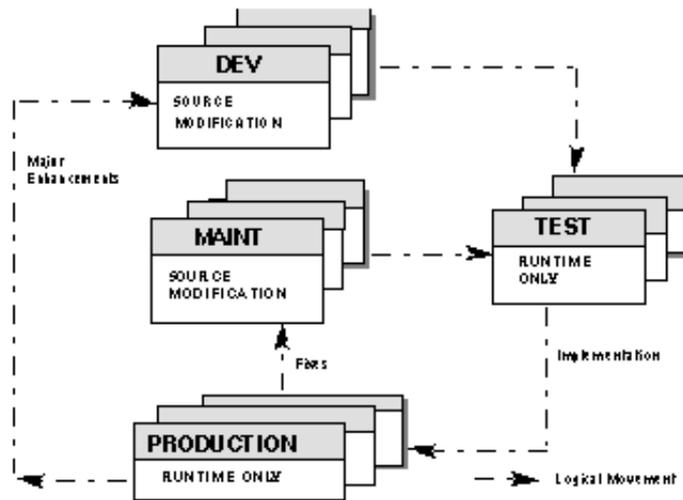
Application Phases

Applications typically move through more or less formal phases, or statuses, of development, testing, and production. A maintenance status with unique functions distinct from those in development may also exist. These statuses are summarized in the following table:

Status	Description
Development	The application source code is developed and/or modified and major enhancements are added to fulfill a specific purpose for the organization.
Testing	Modifications and enhancements are tested before the application is put into production. Types of tests might include <ul style="list-style-type: none"> - Functional - System - Integration
Maintenance	Modifications including minor enhancements to the source code may need to be made apart from development, especially if the development effort is at a different level from that of the code being maintained. If no distinct maintenance phase is recognized, maintenance would be considered part of the development phase.
Production	The application fulfills its purpose within the organization.

Programs are copied to and from locations. These are usually Natural libraries, Predict system files or partitioned datasets (PDSs) for foreign objects. A single application may comprise various objects in a set of locations having a single name. These named location sets are called deployments, and are discussed later under The PAC Controlled Environment.

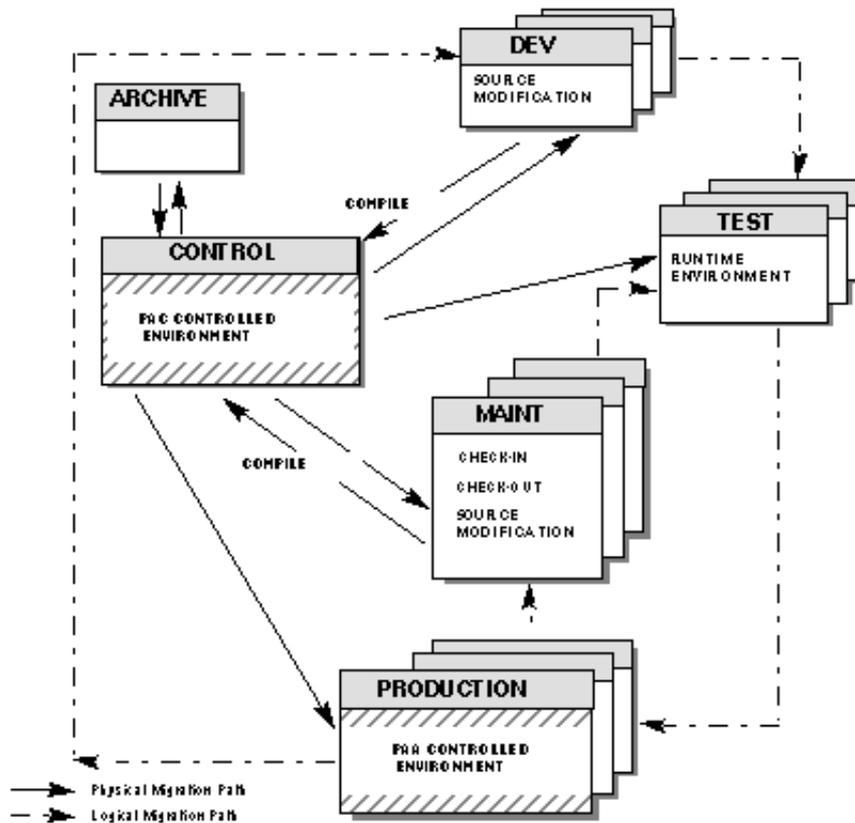
Application statuses are linked by the logical paths typically taken by application objects as they move through the application life-cycle. An Application Environment:



Integrating PAC

You can integrate PAC into your application environment to control the application without significantly changing the way the application moves through its life-cycle. In general, application phases become defined PAC statuses. Application objects are migrated from one status to another.

The following diagram shows the PAC system integrated into the application environment shown above.



The illustration above indicates the physical and logical paths of objects as they are migrated between the PAC statuses in the application life-cycle. Logically, PAC simulates the application life-cycle, but physically, objects are migrated in and out of PAC.

The PAC Controlled Environment

Two files are used to physically implement PAC in your environment: the PCF and ACF system files. All applications defined to PAC are linked automatically to the Control status which is represented physically by these two files.

When Predict Application Audit (PAA) is implemented, the PAA system file is also used.

The Application Control System File (ACF)

The PAC ACF file is a protected file that stores:

- Source and executable code for every object version in the PAC-controlled environment;
- Object-version information;
- PAC entity definitions;
- JCL used for batch PAC jobs;
- System and user profile information; and
- Control data about the PAC installation itself.

The Predict Control System File (PCF)

The PCF file contains the current versions of all objects for all applications under the control of PAC. PAC keyword definitions and generation defaults are also stored on this file.

To guarantee consistency in the PAC controlled environment, Natural objects are saved and compiled here using the current object versions of subordinates as defaults in the compile process. Additionally, Natural rules and views/DDMs are generated here from Predict verifications and userviews.

The file has the same file definition table (FDT) as a Predict file. It is not accessible to a user, but is used as a work area during migrations that involve the development, maintenance, and incorporation statuses.

Tracking the Application's Path

The ACF system file records the state of an application. In most PAC cases, "updating an application" means in fact updating the application's ACF record. Among other things, the ACF tracks the location(s) of the components comprising the application. These combined locations are grouped into a named set called a deployment.

There are five types of deployments:

- Incorporation
- Development
- Test
- Maintenance
- Production

A deployment becomes one of the above types when it is defined in the link between the application and a status. A given location (say, the application's job control in a partitioned dataset) could occur in more than one application-status link; however, it is better to avoid such cases, since marking an application-status link with one of the five deployment types applies to all application-status links for that particular status.

PCF/ACF System File Location

You can decide where to locate the PCF and ACF system files based on the production, development, or test environments. Before you decide, however, you need to consider the security, network, and distributed environments operational procedures (such as backup and recovery) of the location environment.

For example, monthly or even weekly backups may not be adequate because PAC files generally contain "production" data in the development database; that is, objects of applications implemented for production use. On the other hand, placing PAC files in a production database may affect performance due to programmer and quality assurance activities.

When to Implement PAC

Certain physical constraints in your environment may affect the way applications are handled in the PAC (and PAA) systems:

- the responsibilities of users, including those authorized to perform PAC functions;
- the sequence of testing through which the application passes;

- the location of application development, maintenance, testing, and the implementation of the application in production;
- the access to various functions and the movement of objects to various locations according to the environment definition.

Based on the requirements of your environment, you must decide when in its life-cycle you will place your application under PAC control. Two possibilities for implementing PAC are illustrated in the following examples.

Example 1: After the Code is Written

PAC is traditionally implemented into the application environment after the code is written and as the first phase of testing begins.

Advantages: The state or progress of application objects can be tracked during all phases of the application's life-cycle, especially during testing.

Disadvantage: Because the activity of the physical PAC system files is intensified, increased maintenance to these files is required.

Example 2: After the Testing is Completed

Objects may be placed under PAC at a later stage after all testing is completed; the final phase of test and acceptance can then be completed under PAC control.

Advantage: The number of object versions in PAC is reduced. During testing, superseded versions of objects are simply discarded. Only object versions that reach the final testing phase are placed under PAC control and tracked.

Disadvantage: You have less information about the progress of the application; minimal versions of the code are retained.

What PAC Functions to Implement

The PAC architecture allows you to implement its comprehensive set of change management functions and facilities in phases. You can start with basic PAC functions and build expanded PAC capabilities into your environment as and when the need arises.

The first phase includes PAC's basic migration facilities for implementing application(s) from development into production. These migration facilities allow you to

- control the migration process;
- automatically version and track all objects loaded into PAC;
- save time with automated procedures; and
- track all of your activities as you move your application from development into production.

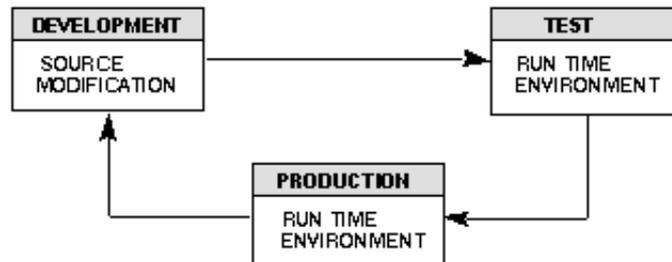
Once you are familiar with the basic PAC migration procedures, you can implement additional PAC capabilities according to your own requirements.

Sample Application Life-Cycles

This section contains some basic application examples. They can serve as a guide as you start using PAC and begin to think about how PAC can best serve your individual application needs.

Sample 1: Basic Setup

The basic setup example provides control over the migration of your application from development to testing, and from testing to production. The basic application life-cycle:



If you test during development, you might eliminate the test status. The application life-cycle would then be:

Development ->Production ->Development

Sample 2: Add a Second Test Status and a Maintenance Status

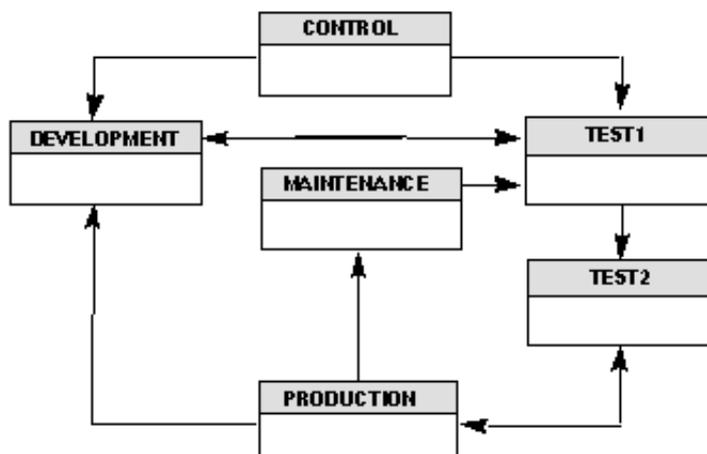
The second example includes an additional test status and a maintenance status.

Additional test statuses can be used if your application testing comprises various phases such as unit, system, and acceptance testing.

The maintenance status accommodates bug fixes and minor enhancements to your production applications. It is separated from development to avoid any conflict with other ongoing development. Maintenance may reflect changes to production while development may contain a later phase of the application that is no longer consistent with the code in production.

The Control status indicates that the application is under the control of PAC. The Control status can be used as an intermediate step when migrating objects; however, if it is not required, migrating directly to the Control status can be deleted from the test plan. When objects are migrated from a development or maintenance environment into PAC, they are always implicitly migrated to the Control status.

The modified life-cycle with test and maintenance phases is represented by the following diagram:



Additionally, you might use an incorporation status for loading objects into PAC initially. The incorporation status allows you to take on an existing application without its objects being recompiled in PAC.

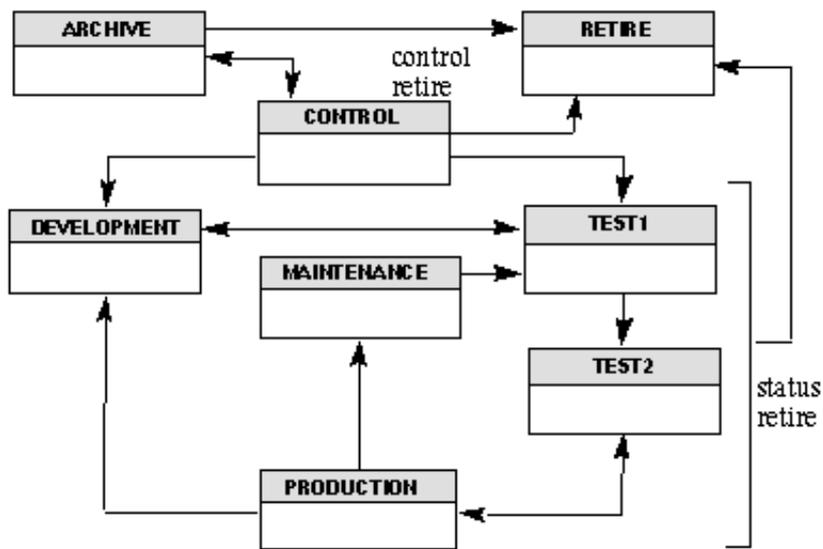
Sample 3: Add the Archive Status and a Retire Status

The third example includes two additional statuses: Archive and Retire.

The Archive status is used to store objects off-line until they are needed again or until they are permanently removed from the PAC system.

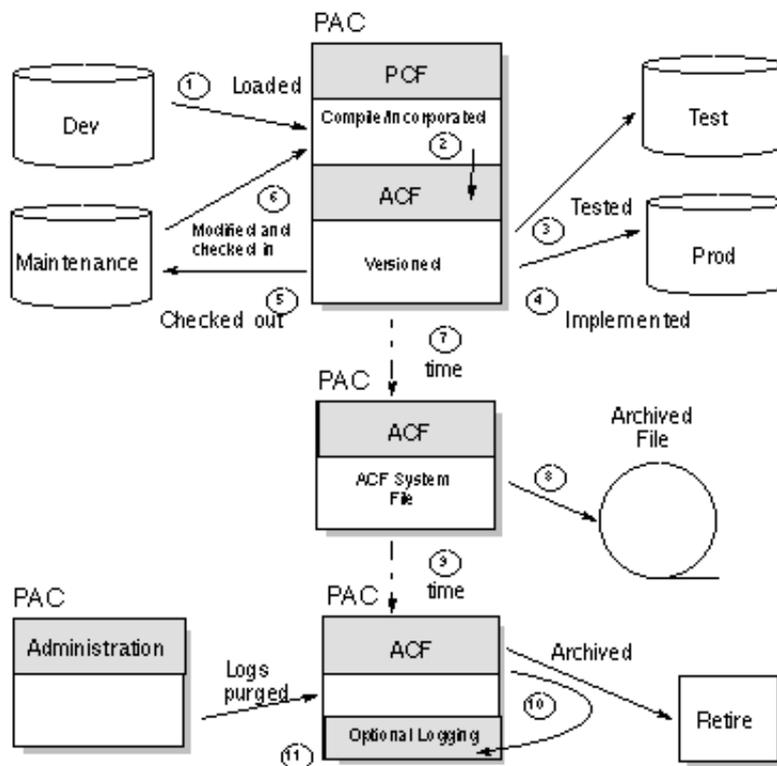
A Retire status may be used to remove an object from a particular status, or out of PAC completely. For example, if an object is migrated to maintenance, and then it is determined that modifications are not needed, it can be "retired" from the maintenance status; that is, the maintenance of the object can be cancelled and the object itself removed from the maintenance environment.

The following diagram shows the life-cycle with archive/retire phases:



Life-Cycle of a PAC Object

Following is an example of a typical life-cycle of an object within PAC illustrating some of the PAC facilities:



The steps illustrated in this diagram are summarized as follows:

Step	Action
1	The object is migrated into PAC.
2	The object is compiled and then versioned.
3	Test the object.
4	Implement the tested object into production.
5	Check the object out for maintenance.
6	Check the modified object back in. The object is then retested and implemented again into production and supersedes the current production version.
7	The superseded object ages.
8	The superseded object is archived and unloaded.
9	The archived object ages.
10	The archived object is retired.
11	If logging of retired (purged) objects has been implemented, the final step is to purge the logs of the purged object and totally remove all audit information for the object from PAC.

Foundations of PAC Control

Three basic principles of change control systems are observed by PAC:

- Objects are compiled only once;
- Only tested code is placed in production; and
- The source code is located in only one place to prevent possible inconsistencies.

Except for foreign objects, objects are compiled by PAC only once for the following reasons:

- To avoid recompiling and all of the effort and resources used to set up an environment with all of the objects to be compiled, including the subordinates (data areas, views, rules, maps, copycode) needed when the object is migrated to a new environment (test or production);
- To prevent anomalies from occurring if objects are recompiled with changed code or with different instances of their subordinates.

For example, if a piece of copycode is modified, it is not necessary for all objects that use it to be recompiled. If the copycode is changed and the object is recompiled, the new version of the object that uses this copycode will be untested and may no longer execute in the same way that the previous version did.

By not recompiling, the PAC methodology of implementing only the tested code is maintained.

The PAC methodology requires that all versions of the code be maintained in a single location so that the latest source code can be located at all times and remains distinct from previous versions of code that have been tested. Programs that use dynamic source variables are the exception to this.

The basic purpose of PAC is to support the integrity of the production environment; therefore, Software AG recommends that source code not be available in the production environment for the following reasons:

- To maintain integrity, PAC must ensure that all objects running in production are synchronized with PAC and have proceeded through the appropriate control and tracking mechanisms inherent in the PAC product and defined by the user (primarily, the migration paths).
- When source code resides in the production environment, changes can be made to it and the code can be recompiled. Such changes are incompatible with the basic tenets of change management.

For foreign objects, PAC assigns versions to both the source and object (loadable) code. PAC does not force a recompilation of these object types; instead, PAC assumes that the source and object code match. Once migrated into PAC, these foreign objects follow the same application life-cycle as Natural-based code.

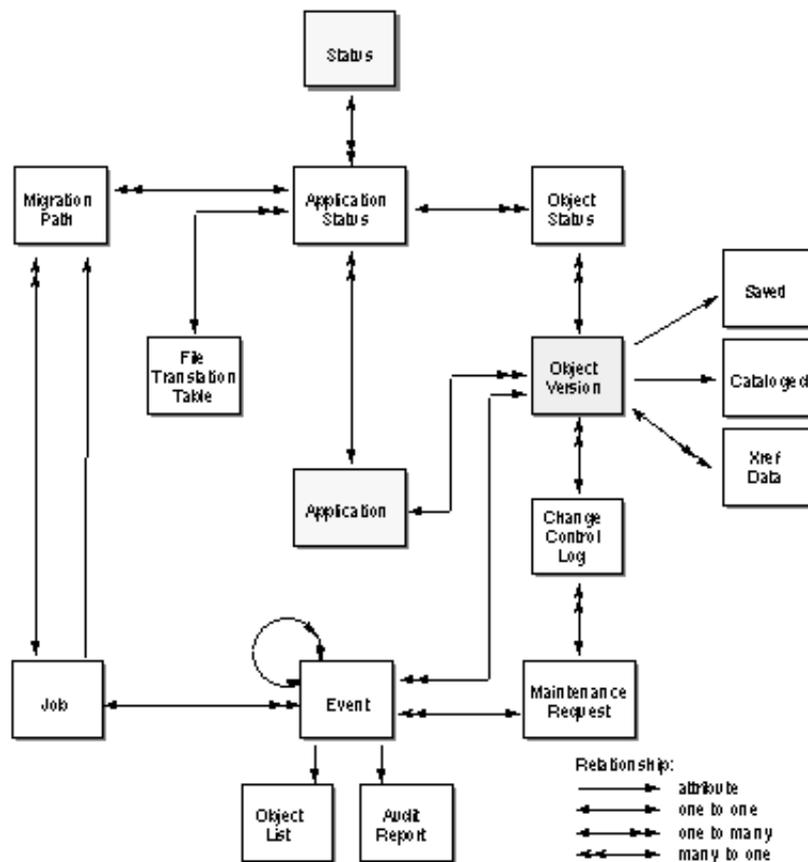
PAC Entities and Their Relationships

An object and any other life-cycle component and path defined to PAC is a PAC entity. The following basic entity types used in the PAC and PAA systems are initially defined to PAC in the following sequence (these and other entity types are described in the glossary):

Order	Entity
1	Status
2	Application
3	Application status link
4	File translation table (FTT)
5	Migration path
6	Job
7	Maintenance request
8	Migration entities (migration event, object list, audit report)
9	Versioned object
10	Change control log
11	Keyword

Once the PAC entities are initially defined, you can add or modify the definitions with minimal concern for the sequence. For instance, entities such as maintenance requests that are not immediately needed can be defined as the need arises.

The relationships among PAC entities are illustrated in the following diagram:



As the individual PAC entities are discussed in the following sections, their immediate relationship to other entities is reiterated.

PAC entities can be identified by optional keywords - user-defined tags for identifying PAC entities or object for reporting purposes. Keywords can identify application components, or the users who own and/or use the entities. Keywords can be user-defined or preassigned by the PAC administrator for automatic assignment whenever a user creates an entity.

For more information on defining keywords, see Appendix C. Information on assigning predefined keywords is contained in the later descriptions on creating the respective entities.

PAC Administration Facilities

PAC provides administrators with facilities for performing system-level maintenance functions as described in the PAC Administration documentation:

- Add, modify, and delete user profiles, including authorization levels.
- Maintain defaults for applications, applymods, system profile descriptions, and Predict generation.
- Activate and deactivate user exits.
- Define foreign support for the PAC installation.
- Release locked data.
- Reset migration events so that they can be rerun.
- Maintain table definitions for maintenance requests.
- Maintain security definitions for views.
- Finalize the process of archiving objects.
- Purge change control logs, audit histories, and obsolete object versions. Object versions can be purged with or without first archiving them.

PAC Utilities

PAC provides three types of utilities: migration, scan, and compare. PAC utilities are described in the PAC Reference documentation.

The Migration Load (MIGLOAD) Utility

The PAC migration load MIGLOAD utility performs the same functions as the Natural utilities NATLOAD, but also creates and uses PAC control information. If you want to use only one utility in a mixed environment, you can use MIGLOAD for non-PAC objects as well.

The MIGLOAD utility loads Natural and foreign saved objects and cataloged objects, views and user error messages. A MIGLOAD report lists the name, type, and version of each object loaded; its destination library, database, and file; whether or not it replaced an existing object with the same name in the destination library; and other information.

You can execute MIGLOAD in either batch or online mode, using menu selections or direct commands.

The Scan Utilities

Scan utilities allow you to scan Natural object versions and migration paths. These utilities can be executed in batch or online.

- SCANOBJ is used to scan for any string of characters in the source code of versioned objects that belong to an application defined to PAC. You can specify a range of objects or object types, a version number, a range of version numbers, or all versions in a particular status. The utility supports absolute scans.
- With SCANPATH, you can scan all migration paths for a specified job name. This utility includes options to replace the job name selectively or globally and to modify migration-path defaults.

The Compare Utility

The PCA compare utility allows you to compare both Natural and foreign object types. Outside of PAC/PAA only Natural objects can be compared; inside of PAC/PAA, both Natural and foreign objects. You can compare:

- Individual objects as well as lists of objects;
- directory information of objects

It is also possible to compare an object outside of PAC/PAA with one that is controlled by PAC/PAA. For a detailed description of the Compare Utility see the PAC Reference documentation.

Facilities for Customizing PAC

PAC provides Application Program Interfaces (APIs), more than 30 user exits, and more than 30 aplymods to help you customize the PAC environment to meet your specific requirements. These facilities are described in detail in the PAC Reference documentation.

Application Program Interface (API) Facility

PAC provides an Application Program Interface (API) facility to simplify the process of customizing PAC to meet your site requirements. The APIs provide flexibility in accessing PAC information and interfacing with your existing systems.

PAC APIs are Natural subprograms that can be used to call PAC directly from within a user-written program without invoking PAC with MENU, ADMIN, or SYSPAC commands. For example, you can create a maintenance request or migration event without entering the PAC system. When you enter information on a front-end screen, the information is communicated to PAC.

The maintenance-request API lets you interface an external problem-tracking system with PAC. You can validate maintenance requests against the external system or extract data from it.

Another API lets you retrieve statistics for an object, display source code for an object version, and edit the source code from the work area.

User Exits

At various points in the system, PAC provides user exits that pass control to a user-written Natural program. When activated by the administrator, these exits allow you to

- add site-dependent code into PAC processing for auditing, security verification, or additional rules;
- add functionality to certain PAC processing; or
- override default change-management procedures.

For example, if the user exit PACEX010 is active, it is invoked during the validation of an object list for a migration. The PAC administrator can view each entry in the list and disallow the migration of specific objects.

Applymods

System applymods change certain PAC or PAA processing defaults. For example:

- The user who authorizes a PAC migration event can normally override some event parameters during authorization. Applymod 1 disallows the override.
- By default, PAC creates a new version of an object even if the object fails to compile successfully during a migration from a development or maintenance status. Applymod 14 rejects an object with compilation errors so that no new version is created.

A user who authorizes an event can activate or deactivate applymods by selecting from a list. The PAC administrator establishes the default applymod settings and can prevent specified settings from being overridden by the event authorizer.