Predict Application Control

Concepts

Version 2.6.1

November 2016

This document applies to Predict Application Control Version 2.6.1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: PAC-CONCEPTS-261-20161120

Table of Contents

1 PAC / PAA Concepts and Facilities

Predict Application Control (PAC) is a flexible tool for controlling Natural and foreign applications throughout the software life-cycle and for ensuring the integrity of applications in the production environment. PAC facilitates and controls the movement of applications through the life-cycle. When an application is implemented into production, it is protected and audited by Predict Application Audit (PAA). You can use PAC and PAA to perform the following functions:

- Develop and enforce site- or application-specific software life-cycles.
- Establish who can promote applications through the life-cycle.
- Control and audit the migration of applications from one location to another.
- Analyze the impact of proposed code changes.
- Track and control maintenance activities.
- Protect the source code for each version of an application component and link it to the executable code.
- Maintain accurate relationships among versions of application components.

This documentation discusses the problems of application control and how PAC and PAA address them. It also describes how PAC and PAA work and the facilities they provide.

This information is organized under the following headings:

2 Introduction

This introduction covers the following topics:

Terms Used in this Document

This section defines basic terms used throughout this documentation. For your convenience, the definitions are repeated in a glossary, along with other terms introduced in later documents.

Objects and Types of Code

An object is one of the following:

- A unit of programming code (for example, a program or subroutine);
- A data-dictionary definition (for example, a view of a database file);
- A user-written error message.

Natural views (which can be generated from Predict views) are called data definition modules (DDMs). To accommodate the terminology's used at different sites; they are called views/DDMs in this documentation.

An object version is an object that has been put under the control of PAC and assigned a version number. An object version is sometimes called a PAC object or versioned object.

Source code means a unit of code written in a high-level language, such as Natural. Source code is equivalent to Natural "saved" code.

Executable code or executable object means a unit of code that has been compiled and can be translated by Natural. Executable code is equivalent to Natural "cataloged" code. A PAC entity is any of the structures that PAC uses to control or facilitate the movement of objects through the life-cycle. These entities are discussed in **Using PAC**.

Applications, Libraries and Locks

An application is a set of objects that work together to perform a task. Thus, an application object is a component of the application.

A Natural library is a set of Natural objects that are stored together; you can add objects to a library or delete objects from it.

A foreign location is the physical location of the dataset that contains the foreign objects.

A lock prohibits access to a library, application, or object.

System Files

System files are Adabas files that contain the following (the name of each system file is in parentheses):

- Software resources (FNAT)
- Application libraries (FUSER)
- Predict information (FDIC)
- Security information (FSEC)

In this documentation, the FUSER file, which contains application objects, is called the user system file. The FDIC file is called the Predict file. In addition to Predict data definitions and cross-reference information, Natural rules and views/DDMs generated from Predict entries are stored in the Predict file. User system files and Predict files are identified in PAC by the database number (DBnr) and file number (Fnr).

In addition, there are three system files reserved to PAC and PAA:

1. The PAC ACF systems file stores the saved and cataloged code for every version of every object under PAC control. The ACF also stores extended information about object versions and PAC entities.
2. The PAC PCF system file stores cross-reference data, keywords, and the latest version of Natural and Predict objects in the PAC-controlled environment. Natural objects are cataloged and Predict objects are generated in the PCF.
3. The PAA system file stores audit data for production objects.

Environments and Directories

The user environment is the mix of hardware, operating systems, teleprocessing (TP) systems, and database managers (DBMSs) at a site. An application environment or operating environment includes the physical locations where application objects are stored, and the operating system, TP system, and DBMS with which they are developed, tested, or used. The PAC-controlled environment refers to the PAC ACF and PCF system files and the operating environments that are governed by site-specific procedures defined in PAC. The PAA-controlled environment refers to production environments that are protected and audited by PAA. A Natural directory provides information about the environment in which an object was developed, saved, or cataloged.

Integration and Requirements

PAC is fully integrated with other Software AG products and at various levels does have certain version requirements. These version requirements are described in detail in the PAC Installation documentation and PAC Release Notes that are issued with each version. Other Software AG products that may be required by PAC are:

- Natural for Mainframes
- Natural Security for Mainframe
- Natural for UNIX / OpenVMS
- Predict
- Adabas
- Entire System Server
- Predict Case

3 Application Control

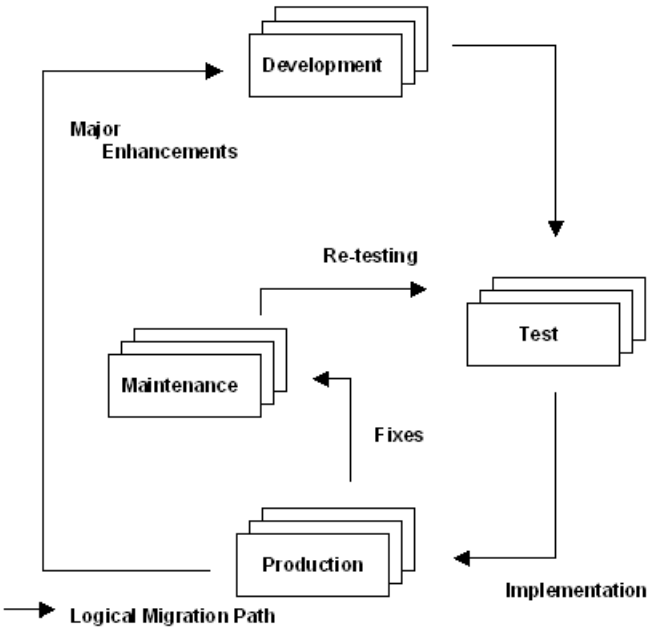
Data processing applications and user environments are continually becoming more complex:

- Multiple hardware and operating system configurations frequently coexist in the same facility.
- Multiple programs may access the same routine in an application library or across applications.
- Several programmers may change the same code concurrently.
- Different versions of a program or routine often exist in development, test, and production libraries.

This increased complexity has multiplied the problems of controlling an application throughout its life-cycle.

- **Inventory Management**
- **Configuration Management**
- **Migration Management**
- **Production Management**
- **Change Management**

The following figure shows an example of an application life-cycle in a heterogeneous user environment.



Application control issues can be broken down into five areas of software management:

1. Inventory

2. Configuration
3. Migration
4. Production
5. Change

The following sections present the questions and processes involved in each area and explain how PAC addresses them.

Inventory Management

This section answers the following questions:

- Where is the source code for an object?
- Do the executable objects match the source code?
- What is the current version of an object?
- What other objects does an object need for successful compilation and execution?
- Which version of an object does a calling program use?
- How are obsolete objects handled?

Managers and auditors increasingly cite the need for better control over source and executable code. Inventory management maintains the source and executable code for all components of an organization's applications. It maintains the source and executable libraries, links executable objects to source code, and provides an efficient way to retrieve the source code.

Each time an object is migrated from a development or maintenance environment to another environment defined to PAC (for example, for testing), PAC compiles the object, assigns it a new version number, and stores both the source and executable code in a protected file. PAC uses a date-time stamp to synchronize the source and executable code for each version of an object, or object version. This mechanism happens automatically for Natural objects, but has to be controlled using supplied APIs for foreign objects.

A reporting facility shows all versions of an object in the PAC-controlled environment, including the current version. For each object version, you can display the source code, directory information about the operating environment in which the source code was developed, and other information.

When the relationships among versions of component objects are not up to date, a new version of an application may fail in production. Inventory management maintains accurate relationships among versions of interdependent objects.

PAC automatically cross-references object versions. Thus, not only can you list all the objects that an object uses or all objects that use the object, but also you can list the specific object versions. This is only possible for Natural objects.

To help you keep track of object versions, PAC enables you to recompile an entire application using the latest versions in the PAC-controlled environment or other versions that you specify.

PAC includes two facilities for handling obsolete objects. When you archive objects, you can unload them to disk or tape and purge them from the PAC-controlled environment, if necessary, you can later restore them to PAC, this is called archiving. When you retire objects, you purge them from the PAC-controlled environment by migrating them to a "virtual" environment; objects may or may not be archived before being retired.

Configuration Management

This section answers the following questions:

- How are the development, test, production, and maintenance environments defined?
- What operating environments are defined for a particular application?
- What is the physical location of a particular application environment?
- What is the life-cycle of the application?
- Who can authorize migrations of objects between two environments?

Configuration management establishes the operating environments for applications. For each environment, it specifies the physical location, how objects are handled, and what applications may use it.

For each application, configuration management defines all the environments the application may use during its life-cycle, the valid migration paths among them, and the controls on those paths. The structure of permitted environments and migration paths constitutes the application life-cycle.

In PAC, you can define the attributes of an application environment once and enforce them for all users. PAC allows you to define multiple development, test, production, and maintenance environments. Each application is assigned to a subset of the environments; applications can share environments.

Managers can control who may define the valid migration paths among each application's environments. Each migration-path definition includes controls on migrations along that path, including who may authorize the migration. A configuration of migration paths can be copied from one application to another.

Thus, PAC provides both central control and flexibility. Managers can establish the degree of central control in configuring operating environments and authorizing migrations among them.

Depending on the needs and philosophy of your organization, you can create varied or uniform application life-cycles.

Migration Management

This section answers the following questions:

- How do you handle migrations when the origin and destination environments use different operating systems, teleprocessing (TP) systems, database management systems (DBMS's), and databases?
- How are procedures for migrating or promoting objects enforced?
- How do you ensure that the correct object version is implemented and referenced by other objects?
- How do you recover when an object is accidentally overlaid?
- How do you handle a situation in which an object was compiled referencing one set of databases and files and you need to execute it referencing a different set?

Control over the migration and promotion of applications through the life-cycle is essential to application integrity. Migration management includes controls to ensure that objects are migrated only to approved locations and that appropriate personnel authorize all migrations. It provides a way to recover from accidental overlays and an accurate audit trail of all migration activities.

When you migrate an object, migration management helps ensure that all objects needed to compile or execute the object are migrated also. For example, assume that Program (A) uses Subroutine (B) during execution; if Program (A) is migrated to a new location for testing, there must be a way to ensure that Subroutine (B) is migrated with it.

PAC operates independently of the operating system, TP system, and DBMS. Thus, you use the same PAC procedures to migrate applications among heterogeneous environments.

Your definitions for the possible phases of an application's life-cycle, the physical location of each phase and the valid paths for migrating objects to each phase enforce procedures. PAC migrates objects only in accordance with your site- or application-specific definitions.

When you migrate or promote an application to another phase in the life-cycle, PAC uses the latest object versions in the library by default; you can override the default and specify a different version of an object. You can display the directory or source code for any object version or compare versions of an object.

A protected PAC system file stores a copy of every version of every object in the PAC-controlled environment. If one of these versions is accidentally overlaid in a library, it can be reloaded. In addition, when you replace a production object with a new version, PAA automatically backs up the old version; you can restore the earlier version from the backup.

PAC file translation tables save time and computer resources by allowing you to execute compiled objects against different files and databases without changing and recompiling the source code. When an object referencing user data on one set of databases and files is migrated for execution referencing user data on a different set, PAC dynamically recompiles the object to reference the new database and file numbers.

PAC includes a migration utility with several modes for selecting objects. PAC maintains an audit trail of all migration activities.

Production Management

This section answers the following questions:

- How is implementation of application objects controlled?
- When there are multiple production environments, how do you know where objects are implemented?
- How is implementation tracked?
- What facilities exist for recovery when a newly implemented object causes problems in the production environment?

The ultimate goal of any application-control system is to ensure the integrity of applications in the production environment. An organization must be able to protect its production code. Production management protects, audits, and controls objects in production environments.

Although inventory, configuration, and migration management help ensure that correct and tested versions of objects are implemented into production, an application may still cause a system failure when implemented. When this happens, there must be a way to back out quickly to an earlier version of the application or objects.

Predict Application Audit (PAA) protects each production environment and audits all migrations of objects into the environment. When an application is migrated to production, PAA backs up the existing production version of any object that will be replaced by a new version. The PAA administrator can back out a migration, which automatically restores the previous versions of objects that were replaced by the migration.

PAC and PAA also enable you to migrate applications to production in advance and schedule the earliest date and time that they can be activated. The PAA administrator controls the activation.

PAA provides extensive reporting facilities. You can display information about production jobs, applications, libraries, and object backups. You can list all the applications in a library or all the libraries that contain objects of a specified application.

Auditors will appreciate the ease with which changes to production applications (for example, when versions of objects are superseded) can be detailed. Audit reports provide information about

each object version and its migration to the production environment. PAA also supply reporting APIs for the end-user's use.

Change Management

This section answers the following questions:

- How are maintenance activities tracked and controlled?
- Who is changing an object?
- How are concurrent changes handled?
- How are related problems grouped?

Change management coordinates the maintenance activities of programmers, ensures that critical problems are resolved quickly, and minimizes the time required. It includes facilities to do the following:

- control updates to code;
- enforce standard maintenance procedures;
- track maintenance requests and activities;
- report the status or history of changes;
- analyze the impact of proposed code changes.

In PAC, when an object is migrated to a maintenance environment, a check out facility automatically logs detailed information about the migration, including the identity of the user who checked the object out. PAC logs the same information when the object is checked in (migrated from the maintenance environment). Users can display or print the logs. PAC controls migration to maintenance environments the way it controls migrations or promotions to any phase of the life-cycle: objects can be migrated only in authorized migrations along defined paths.

You can monitor and prohibit concurrent maintenance activities; by default, PAC handles them with a warning. For example, assume that Programmer A has checked Version 1 of the DISPLAY-ORDER subroutine out for maintenance and has not yet checked it back in; when Programmer B checks out this version of the subroutine, PAC issues a warning to Programmer B. Alternatively, the PAC administrator can monitor the check-outs and block Programmer B from checking out Version 1 until Programmer A checks it back in.

Each time an object is checked back in from a maintenance environment, PAC gives it a new version number. Using the previous example, assume that Programmers A and B concurrently check out Version 1 to maintenance. Programmer A checks the subroutine back in; PAC compiles it and designates it Version 2. Then Programmer B checks the subroutine in; PAC compiles it and designates it Version 3. A PAC utility enables you to compare the two versions.

You can use PAC maintenance requests to document problems with objects, track maintenance activities performed in response to a request, and group related problems and activities. A maintenance request can be linked to an external problem-tracking system.

PAC supports impact analysis by listing all application objects that will be affected by changes to an object.

4 Using PAC

PAC controls the application life-cycle in the following ways:

- Secures the locations of libraries and system files
- Controls migrations and promotions of objects
- Protects source code
- Migrates code from a protected environment
- Ensures that object relationships are up to date
- Tracks maintenance activities

This section is organized in the following topics:

Overview

An application goes through a series of phases in its life-cycle. These phases typically include development, testing, implementation into production, and maintenance.

PAC allows you to monitor and control an application's life-cycle by defining a series of statuses through which it is promoted. Each status defines a phase in the application's life-cycle and the environment in which the application functions in that phase.

For example, an application might be brought into the PAC-controlled environment from a development status, move through several test statuses, and be implemented into a production status. When changes to the application are needed, the affected objects are copied to a maintenance status; after the changes are made, the objects might again move through one or more test statuses and back into a production status.

An application is promoted from one status to another through an online or batch migration. The migration process includes the following:

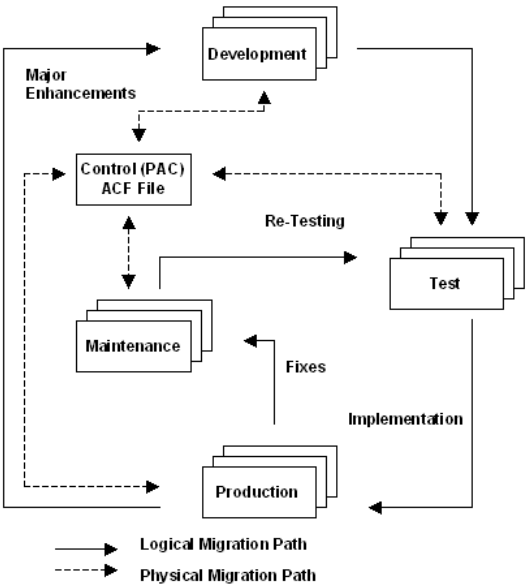
- defined origin and destination statuses,
- a list of the objects to be migrated,
- a scheduled migration event,
- job control statements (for batch migrations),
- an audit of the migration activities.

Controls and facilities are built into the migration process to ensure that the application is migrated to the correct location, the set of objects to be migrated is complete and correct, and the migration is made with the approval of the appropriate authority.

A logical migration is a promotion to the next status in the life-cycle; it may or may not include a physical migration of objects. A migration to a new physical location is both logical and physical;

the logical and physical paths may differ. When two statuses share the same physical location, a migration from one status to the other is merely logical. Thus, all migrations have logical paths, but not all have physical paths.

Logical paths can be defined according to the needs of your site and applications. However, to help you maintain the integrity of applications at every stage of the life-cycle, the code is always physically migrated to and from the protected PAC ACF system file. The following figure shows how PAC controls the typical software life-cycle.



PAC Entities

Application

An application is defined to PAC by its name and level. It provides default information to be used when objects belonging to the application are processed. The objects that compose the application may change as the application is developed and tested.

Status and Application Status Link

A site may require multiple testing and production environments. To enable an organization to define a set of statuses and use different combinations of them to control a variety of applications, PAC uses two entities, the status and the application status link.

Status

The type of status determines how PAC handles objects in that status. The following table shows the four basic status types in PAC:

Status Type	Associated PAC Behavior
Development	Source code for objects can be modified freely. PAC recompiles each object migrated from a development status and assigns the object a new version number.
External	The behaviour of this status is a mixture of the behaviour of the statuses Development and Incorporation in the following sense: objects to be transferred will be compiled newly and a new PAC version of the object code is created, but it is not possible to transfer objects into such a status.
Test	Except for objects that use dynamic source variables, PAC migrates only executable code to a test status. Test statuses are unprotected by PAC; you can modify an application by migrating objects with Natural or PAC utilities. However, unless you use a PAC event to migrate the objects, PAC ignores them when it migrates the application to the next status; this restriction ensures the consistency of the application.
Production	All objects are protected by PAA and cannot be modified. Except for objects that use dynamic source variables, PAC migrates only executable code to a production status.
Maintenance	Source code can be modified freely. The PAC check-out/check-in facility tracks objects migrated to or from a maintenance status. PAC recompiles each object migrated from a maintenance status and assigns the object a new version number.

You can define multiple statuses for each basic type. For example, test statuses might include Systems Test, Integration Test, and User Test.

In addition to the basic types, PAC has the four special (PAC) status types shown in the table below. Source code for objects in these statuses cannot be changed.

Status Type	Associated PAC Behavior
Control	Files associated with the Control status contain the source and executable code for every version of every object under PAC control. Each application defined to PAC remains in the Control status even as the application moves through its life-cycle.
Archive	When an object is migrated to the Archive status, it is removed from all other statuses. The object can be unloaded to disk or tape and purged from the PAC system. Active production objects cannot be archived.
Incorporation	An incorporation status is used to bring existing production applications into PAC without recompiling the code.
Retire	A retire status is a logical or "virtual" status used to purge objects from a specified status (including the Control status). Inactive or obsolete objects, including historical objects that have been archived, are good candidates for a migration to a retire status.

Application Status Link

Before it can be migrated or promoted to a status, an application must be linked to the status. The application status link specifies the application, the status, and the physical location where the application's objects are stored at that status:

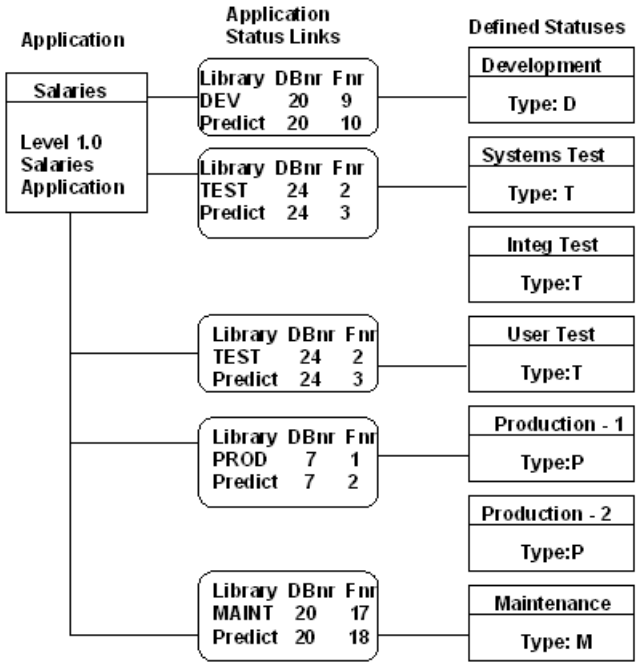
- The library and user system file for Natural programming objects,
- the dataset name and location for foreign objects,
- the Predict file for Xref data, rules, and views/DDMs.

The link definition may also specify a file translation table to be used in migrations to the status. The section File Translation Table later in this section describes the function of these tables.

For certain status types, the link definition may specify a step library to be used when PAC compiles objects during a migration to another status.

An application or object can exist in several statuses at the same time. Thus, an object in a production status could be copied to a maintenance status for changes while the original remains available to users.

The following figure shows an application linked to a subset of the defined statuses. It shows how each application status link defines the Natural library and Predict file for the application's objects in that status. Note that the links to Systems Test and User Test share a Natural library and Predict file.



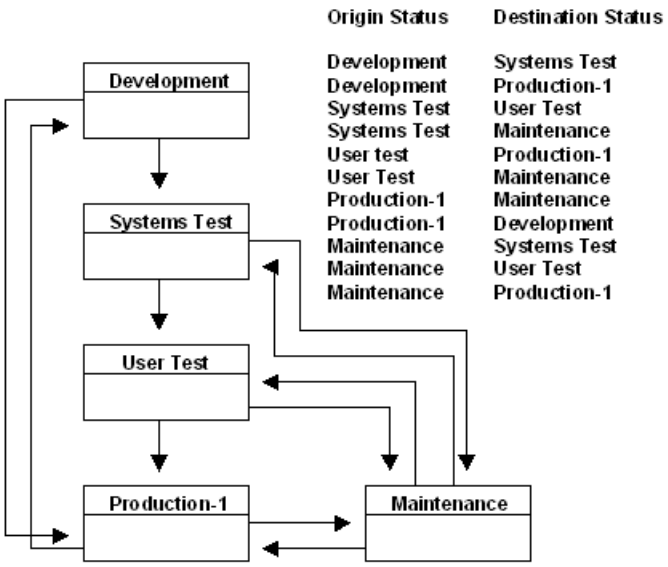
Migrations

As an application moves through its test plan or life-cycle, PAC uses defined procedures and controls to migrate it from one linked status to another. These procedures and controls are defined in the migration path and migration event.

Migration Path

A migration path specifies the origin and destination statuses of the migration. The origin status is the status from which the objects are migrated. The destination status is the status to which they are migrated.

A set of migration paths is application-specific. Before a migration path can be defined for an application, the application must be linked to the origin and destination statuses. The following figure shows migration paths defined for the application SALARIES.



Each migration path also defines the following defaults for migration events along that path:

- The migration mode:
  - Copy maintains the object version in the origin status and stores a copy in the destination status.
  - Move removes the object version from the origin status and stores it in the destination status.
  - Include copies objects from an external work file and stores them in the destination status (used when migrating objects into the PAC-controlled environment).
- The processing mode (batch or online).
- The Auto Expansion option, which specifies whether PAC automatically includes related objects in the list of objects to be migrated and how PAC identifies which versions of the objects to include.
- The PAC job, which contains job control statements for batch migrations.
- The Workfile Usage option, which specifies whether PAC migrates objects directly from one status to another or uses a work file in an intermediate step.

These defaults can be overridden when the migration event is authorized.

Migration Event

A migration event executes the migration. An event can migrate objects only along a defined migration path; this restriction controls the locations to which users can migrate objects.

When defining the migration event, you can select from a list of valid paths. The event can be scheduled for immediate or future processing, online or in batch. When authorizing a migration event, you can set system Applymods, which override PAC default processing procedures. For more information, refer to the section **Applymods**.

Object List

The migration event includes the object list of objects to be migrated, which may include an entire application or a single object. You can create the object list manually, or PAC can generate it automatically. PAC can also add related groups of objects automatically. See the sections `Expanding an Object List` and `Generating an Object List` for information about how PAC automates the task of creating object lists.

Enforcing Migration Procedures

PAC builds several levels of control into the migration path and migration event. Before a migration event can be processed, it must be authorized. Users who may authorize a migration along a path are specified in the migration-path definition; you can specify a single user as the authorizer, a range of users (for example, user IDs that begin with a specified set of characters), or a group ID to which individual user IDs are defined. You can also specify that the user who authorizes a migration event along the path must be different from the one who creates the event.

Similarly, an organization can restrict the authority to define migration paths. In a highly centralized organization, an administrator might be the only one authorized to define migration paths; thus, the administrator would define the migration paths for all applications at the site. In another organization, the project leader for each application might be authorized to define the migration paths for the application. By restricting the authority to define migration paths and authorize migration events, management can control migrations to specific environments.

Control Status and Migrations

Every application defined to PAC is automatically linked to the Control status, and every object migrated into PAC is migrated to Control. The Control status is associated with the protected PAC ACF system file, which stores the following:

- source and executable code for every object version in the PAC-controlled environment,
- object-version information,
- migration information.

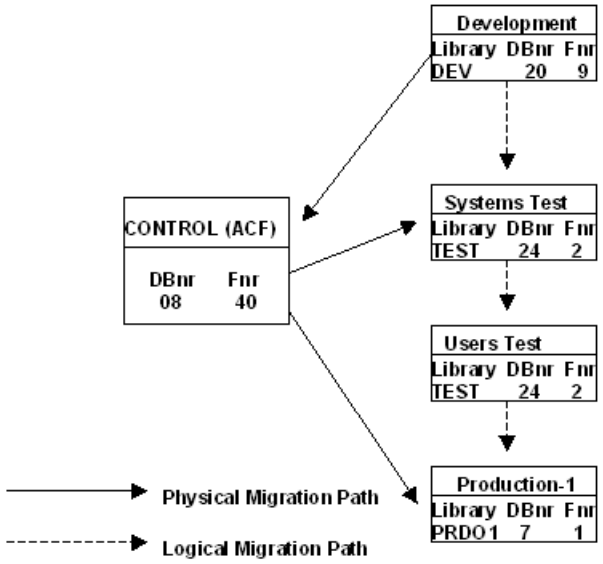
To ensure the integrity of applications at each stage of the life-cycle, the Control status and the ACF play a role in every PAC migration. When an object is migrated from a development or maintenance status, PAC compiles the object and stores the source and executable code in the ACF, along with information about the version and migration. If the destination status is a status other than Control, PAC then copies the executable object to the destination library.

Whenever a migration involves the physical movement of code, the objects are always migrated to the destination status from the ACF, even when Control is not defined as the origin status in the migration path. PAC copies the objects from the ACF into the library at the destination status and records the migration.

When a migration involves no change in physical location (that is, when the migration is a logical promotion only), PAC simply records the migration. The following figure shows logical and physical migrations of an application among five statuses to which it is linked.

The logical migration from Development to Systems Test involves an implicit migration to Control. Although Systems Test is the destination status in the migration event, the code is first compiled and stored in Control. Then it is copied to Systems Test from Control.

The Control Status and Migrations:



The table below describes the steps PAC performs to execute the migrations shown in the figure above:

Migration	Step	PAC Action
Development to Systems Test	1	Copies the objects from the library DEV.
	2	Compiles the objects and stores them, along with version information, in the ACF.
	3	Copies the executable objects from the ACF to the library TEST.
	4	Records the migration to Systems Test.
Systems Test to User Test	1	Records the migration to User Test.
User Test to Production-1	1	Copies the executable objects from the ACF to the library PROD1.
	2	Records the migration to Production-1.

An explicit migration to Control names Control as the destination status in the migration path. The Control status can be used as a staging area and a way to distribute authority. For example, when several programmers are responsible for different parts of an application, you might allow each programmer to authorize migrations to Control, but only the project leader to authorize a migration from Control to Systems Test. The programmers could independently migrate their parts of the application to Control. When all the parts have been compiled and stored in the ACF, the project leader could authorize an event migrating the entire application to Systems Test.

Versioned Object

PAC assigns a new version number to an object each time the object is migrated from a development or maintenance status. For a specific version of an object in a specific status, you can view the date and time it was saved and cataloged, the event that migrated it to the status, other statuses in which the version exists, and related objects.

The object audit history shows every version of the object, the statuses to which each version has been migrated, and the dates and times of the migrations. You can view the source code for an object version.

You can use a PAC utility to compare objects that are either PAC controlled or objects that are not under the control of PAC. Objects can be compared on an individual basis, as well as on a mass basis. Natural, DDM's as well as Foreign objects can be compared. Results of this compare utility can be printed off or even stored in a work file for later use. Full details of this utility can be found in the PAC documentation.

File Translation Table

A PAC file translation table (FTT) enables you to execute applications against different databases and files without changing or recompiling the source code.

Normally, when the source code for an object is compiled, it references database and file numbers in the development environment. When the object is migrated to a location where the user databases and files have different numbers, the source code must be changed to reference the new numbers and then be recompiled. This step not only requires time and computer resources but can also introduce inconsistencies into the code.

In PAC, an FTT is assigned to an application status link. When you migrate an object to the status, PAC copies the object from the ACF and dynamically recompiles it, substituting the correct database and file numbers. Since PAC copies the executable object from the ACF without removing it, neither the source code nor the executable code in the ACF is affected.

The following tables show excerpts from two File Translation Tables (FTTs) for an application (status names are added for clarification). Note that in both tables, the origin database number (DBnr) and file number (Fnr) are those of the Development status; the numbers are always translated from the numbers referenced in the compiled objects.

Origin Status	DBnr	Fnr	Destination Status	DBnr	Fnr
Development	20	9	Systems Test	24	2
	20	10		24	3
	20	11		24	4

Origin Status	DBnr	Fnr	Destination Status	DBnr	Fnr
Development	20	9	Systems Test	27	1
	20	10		27	2
	20	11		27	3

In the Development status, the source code for the application objects references file numbers 9, 10, and 11 in DBnr 20. When an object is migrated from Development to Systems Test, PAC compiles it using the same numbers and stores it in the ACF (as discussed in the section The Control Status and Migrations).

PAC then copies the executable object from the ACF and dynamically recompiles it, changing DBnr 20 and file numbers 9, 10, and 11 to DBnr 24 and file numbers 2, 3, and 4, respectively. PAC stores the recompiled object in the application library at the Systems Test status.

When the object is migrated from Systems Test to User Test, PAC again copies the executable object from the ACF; dynamically recompiles it to reference file numbers 1, 2, and 3 in DBnr 27; and stores the recompiled object in the application library at the User Test status. Although Systems Test is the origin status in the migration, PAC translates the DBnr and file numbers referenced in Development to those referenced in User Test.

You can specify that an FTT is available for all applications with either a Test or Production status,

, or you can restrict it to a specific application and/or status. By restricting an FTT, you prevent it from being applied to objects in unauthorized locations. For example, it might be necessary to prevent access to production data from anywhere but the production environment.

Change Control Log

The check-out/check-in facility is activated automatically by a migration to or from a maintenance status. When an event migrating an object to a maintenance status is processed, PAC creates a change control log for the check-out, which records the following information:

- Application name,
- Object name and version,
- User ID of the user checking it out,
- Terminal ID or batch ID,
- Date and time of the check-out,
- Library and user system file to which the object is migrated,
- Maintenance request ID (if applicable),
- Optional notes by the user.

PAC logs the same information, including the new version number, when the object is checked in from the maintenance status.

PAC audits check-out and check-in activities. A user exit lets you view the audit information, validate the user and object, and disallow the check-out or check-in. The user exit also lets you track and control concurrent maintenance activities. When a user checks an object out, you can list other users who currently have the same object checked out. When a user checks an object back in, you can list other users who still have the object checked out, or who checked the object in after this user checked it out. This mechanism currently only exists for Natural objects.

Maintenance Request

Maintenance requests are an optional PAC feature for documenting problems and maintenance activities. In addition to describing the problem and related activities, a maintenance request can list all objects required to resolve the problem; PAC can then generate an object list automatically from the list in the maintenance request. You can define and assign site-specific codes to indicate the state of a maintenance request (for example, open or closed) and the action to be taken in response. You can also prioritize requests.

When you assign a maintenance request to an event migrating an object to or from maintenance, PAC automatically assigns the request to the change control log. You can select and view change control logs related to the maintenance request.

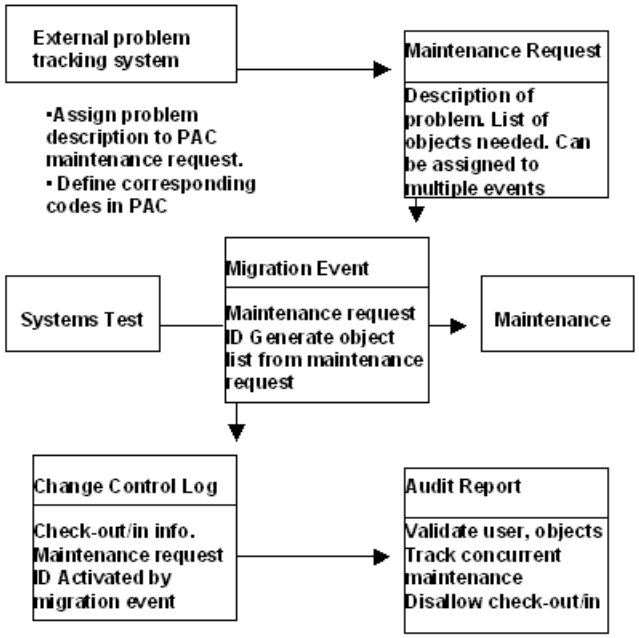
Maintenance requests can be used to integrate PAC with a user-developed problem-tracking system. You can define the PAC maintenance request ID to match the problem ID in the external problem-tracking system. The problem description in the external system can be assigned as an attribute to the PAC maintenance request. The codes for PAC problem states, priorities, and actions can be defined to correspond with codes in the external system.

As related problems emerge, they can be added to a request. You can use maintenance requests to group related problems in two ways:

By defining related problems in a single maintenance request and assigning the request ID to multiple migrations, you can view all migration events related to a problem or assign them to a group migration event. This concept is useful for grouping together all changes that made up a certain release level of an application.

To document complex problems and activities, you can group requests, each of which may be assigned to multiple migrations.

The following figure shows the relationships among PAC migrations to and from maintenance change control logs and audits, and maintenance requests.



With PAC user exits and application program interfaces, the system shown above can be tailored precisely to user requirements.

5 PAC Facilities

This section describes PAC facilities for customizing the PAC system and for expediting your work with PAC entities and objects. It also summarizes the PAC reporting system and administrator functions.

Customizing PAC

Application Program Interfaces

A PAC / PAA application program interface (API) lets you call a PAC function from a user-written program without logging on to PAC and invoking the function from within PAC / PAA. APIs provide flexibility in accessing PAC information and interfacing with your existing systems.

For example, you can create a maintenance request or migration event without entering the PAC / PAA system. When you enter information on a front-end screen, the information is communicated to PAC / PAA.

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.

PAA delivers APIs specifically for reporting use. These APIs enable the user to design and customize user reports based on PAA (Production) information.

PAC includes APIs for working with the following entities:

- Application
- Status
- Application status link
- Migration path
- Migration event
- Object list for an event
- Event authorization
- Audit report for an event
- Maintenance request
- Object version (displaying information, loading Natural source code)
- Automatic compilation of foreign objects
- PAA entity reporting

PAC Express

PAC Express is a delivered EXAMPLE application using the APIs and demonstrates the use of such APIs. It tries to simplify the presentation of PAC to users and reduces the number of actions required to execute a PAC function. PAC Express integrates the PAC APIs to provide simplified screens from which you can perform PAC functions without entering the PAC system.

PAC Express can reduce by half the number of actions required to execute some functions, such as adding, authorizing, and submitting migration events. The modifiable source code is provided so that you can tailor PAC Express precisely to the requirements at your site.

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 do the following:

- Add site-dependent code into PAC processing for auditing, security verification, or additional rules,
- Add functionality to certain PAC processing,
- 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.

More than 30 user exits are available in PAC and PAA.

Applymods

System applymods change certain PAC or PAA processing defaults. For example, by default the authorizer of a PAC migration event can override some event parameters during authorization. When Applymod 1 is active, the parameters cannot be overridden during authorization.

Another example is that 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. When Applymod 14 is active, PAC rejects an object with compilation errors; no new version is created.

The authorizer of 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.

There are more than 30 system applymods for customizing PAC and PAA.

Applications

To define a new application, copy the following elements from an existing application:

- the base application definition (name, level, and processing defaults),
- the base application definition and its status links,
- the base application definition, its status links, and their associated migration paths.

This facility simplifies the process of defining each application and its life-cycle to PAC. Once the first application is defined, others can be defined quickly by copying and modifying the first definition.

Incorporation

Incorporation allows you to bring objects already in production into PAC without recompiling them. This facility is useful when implementing PAC at a site with a large number of application objects in production. An incorporated application establishes the base level for subsequent development, testing, implementation, and maintenance in the PAC-controlled environment.

To incorporate an application, you first define an incorporation status and link the application to the status, specifying the location of the application objects in the application status link. When you migrate objects from the incorporation status to the Control status, PAC assigns them version numbers. Objects can be incorporated directly from a Natural library or using the Include mode in the migration event, you can incorporate objects from a work file.

Predict Case Support

PAC supports Predict Case applications. PAC verifies each object against the Predict Case library definition. Once an application is under PAC control, PAC assigns version numbers and builds cross-reference information about the relationships among the Natural and Predict Case objects.

Object Versions

Aligning Versions

Alignment is a Control-to-Control migration, which recompiles application objects so that they all reference the same versions of subordinate objects. When it recompiles the objects, it assigns them new version numbers.

By default, alignment uses the most recent version of each object. However, you can override the default and instruct PAC to use a different version by specifying the version number or the status in which the preferred version exists.

Aligning an application has the following benefits:

- It simplifies subsequent maintenance.
- It helps you keep track of object versions and dependencies.
- It prevents anomalies (which can occur, for example, when a program that uses copy code is not recompiled after the copy code is changed).

Using the Generate List function, you can also align the objects of an application in two statuses so that the same object versions exist in each status. For more information, refer to the section **Generating an Object List**.

Archiving Obsolete Versions

The PAC archiving facility unloads inactive object versions to an external storage device and purges the physical object from the PAC-controlled environment, leaving only a control record. The unload is performed in batch. Object versions can be restored if necessary.

PAC includes automatic selection of object versions for archiving. The automatic selection is based on the following site-specified criteria:

- The number of object versions in the application
- The age of the object version
- The status in which the object version exists

A user exit can be used to override default criteria for selected objects.

You can use a reporting routine to write the results to a work file for subsequent editing or processing.

Using a batch command, you can dynamically create multiple migration events, including object lists, to archive a range of applications.

Cross-Reference Data

PAC supports Predict Xref (cross-reference) data for objects. The Xref data is stored in the Predict file specified in the application status link. Xref data is only available for Natural and Predict objects and not for foreign objects.

Xref data is useful in nearly all phases of the application life-cycle. It can assist in finding errors and inconsistencies in objects, analyzing the impact of proposed changes, and identifying existing resources that can be used in an application.

The following is a partial list of Xref information that can be retrieved for an object in a specified status:

- External objects invoked by the object
- Methods used to invoke Natural objects (for example, CALLNAT)
- Entry points or functions that can be invoked by other objects
- Copy code used to compile the object
- Maps and help routines used by the object
- Variables and data areas used by the object
- Files and fields used by the object and the type of file/field use (for example, reading or updating)
- Natural error messages used by the object
- Processing rules used in a map
- Work files used by the object

For more information, refer to the Predict Reference Documentation.

PAC automatically cross-references object versions. Thus, you can view the specific version of an object that is used by another object version. PAC cross-reference data for object versions is called object usage data.

Support for Dynamic Source Variables

The source code for some programs is compiled at execution time using the current values of dynamic source variables. These programs must be RUN; therefore, the source code must be migrated to environments where the programs are executed.

PAC and PAA support applications using dynamic source variables by allowing you to migrate Natural source code to a test or production environment. All functionality of cataloged objects (for example, scheduling and backout) is available for source code in production.

Object Lists

Expanding an Object List

The Expand function helps automate the creation of object lists. PAC "expands" objects in an object list by automatically adding specified sets of related objects to the list. Expand can produce the following sets:

- all objects needed to compile a specified object (for example, data areas, copy code, views/DDMs, and maps),
- all objects referenced by a specified object during execution (for example, programs, subroutines, and maps),
- all objects that use or reference the specified object.

Before an object list can be expanded, it must contain at least one object. An object list can be expanded manually or automatically:

When you invoke the Expand function from within the object-list editor, PAC expands each object in the object list.

You can use a selection list to add one or more objects to an object list and instruct PAC to expand the object(s) at the same time.

When defining a migration path or authorizing a migration event, you can instruct PAC to expand the object list automatically during the migration. In the authorization, you can specify whether to expand all the objects in the list or only selected objects.

Even without knowing the version numbers, you can control the versions of objects that PAC expansion adds to an object list. You can instruct PAC to use specific versions of objects, versions in a specified status, or the latest versions in the PAC-controlled environment.

Generating an Object List

The PAC Generate List facility generates an object list automatically from one of the following sources:

- PAC generate, objects selected from origin status, based on name and type
- a Predict set,
- a Predict Case set,
- the list of objects in a maintenance request,
- user-defined archive selection criteria (for archive events),
- input from a work file (batch only).

You can also use Generate List to align the application objects in two statuses. PAC compares the objects in the origin and destination statuses; it includes in the object list only the objects in the origin status that are not in the destination status or that have a different date-time stamp.

Reporting System

You can display and print reports of any PAC entity. Some of the information that can be displayed in PAC is listed below:

- Applications:
  - the application definition (including processing options, general defaults, and default libraries),
  - statuses to which a specified application is linked,
  - applications that have not been linked to any status.
- Statuses:
  - applications linked to a specified status,
  - statuses to which no applications have been linked.
- Application status links:
  - the Natural library and file assignment for the application at the status,
  - the Predict Case library and file assignment (if applicable),
  - the file translation table and step library assigned to the link,
  - defined destination statuses (migration paths) from the status,
  - object versions in the application library at the status,
  - Predict Xref data for the application at the status.
- Migration paths:
  - the configuration of migration paths for a specified application,
  - the authorizers and job for a specified migration path.
- Migration events:
  - the origin and destination statuses (migration path),
  - the Schedule Date,
  - the maintenance request (if applicable),
  - the object list,
  - authorization information,
  - job information,
  - the audit report of the processing of the event.

- Object versions:
  - control and directory information,
  - source code,
  - object versions used by the object at compile time or run time,
  - object versions that use the object at compile time or run time,
  - the status where the object version resides,
  - an object history (all PAC versions of each object for an application, including the statuses where they currently reside).
- File translation tables,
- Jobs (you can display the JCL/JCS),
- Maintenance requests,
- Change control logs.

For information about PAA reports, refer to **Managing and Auditing the Production Environment**.

PAC reports for applications, jobs, object versions, statuses, migration events, and maintenance requests can be customized for your site.

Administrator Functions

PAC provides administrators with facilities to perform the following control and maintenance functions:

- Add, modify, and delete user profiles, including authorization levels.
- Establish defaults for the PAC system, profiles, applymods, descriptions, applications, and Predict generation
- Activate and deactivate user exits.
- Release locked data.
- Reset migration events so that they can be rerun.
- Maintain tables of codes for maintenance requests.
- Foreign table maintenance.
- Maintain security definitions for views.
- Archive event maintenance.
- Purge change control logs, audit histories, and obsolete object versions. You can purge object versions with or without first archiving them.

Utilities

Compare Utility

The PAC Compare utility allows you to compare the following:

- The Natural directories for two objects or two versions of an object,
- Lists of saved (source) and cataloged (executable) objects in two libraries or in a library or PAC status,
- the source code for two objects or two versions of an object,
- the contents of two Natural libraries,
- foreign objects under the control of PAC,
- PAA controlled objects.
- In addition to the Compare, the results can be written away to a workfile, printed, saved to a Natural text member or manipulated in several different ways.

You can compare two PAC object versions, two non-PAC objects (objects in Natural libraries outside the PAC-controlled environment), or a PAC object with a non-PAC object. Compared objects can reside in different libraries.

To identify an object version for comparison, you can specify either the version number or the status in which it resides.

Migration Utilities

The PAC utilities MIGRATE LOAD (MIGLOAD) and MIGRATE UNLOAD (MIGUNLD) perform the same functions as the Natural utilities NATLOAD and NATUNLD but also create and use PAC control information. If you want to use only one set of utilities in a mixed environment, you can use MIGLOAD and MIGUNLD for non-PAC objects as well.

With the MIGUNLD utility, you can unload objects from a set created using Predict Xref data, objects from Natural libraries, and Natural views/DDMs from a Predict file. For each unloaded object, the MIGUNLD report lists the object name and type, version, destination library, origin database and file numbers, and other information.

With the MIGLOAD utility, you can load Natural saved objects, 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 it replaced an existing object with the same name in the destination library; and other information.

You can execute MIGUNLD and MIGLOAD in batch or online, using menu selections or direct commands.

Scan Utilities

PAC includes scan utilities:

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.

With SCANOBJ, you can scan a PAC application's Natural objects for a character string. 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.

6 Managing and Auditing the Production Environment

This section is organized in the following topics:

The production environment is the culmination and purpose of the application life-cycle. The integrity of this environment is critical.

The nature of development and testing environments requires that they be administered flexibly. A wide range of people may be involved in controlling access to applications at these stages of the life-cycle.

However, the production environment requires a high degree of control, along with special procedures to ensure the timeliness of implementation. For this reason, PAC views each production environment as distinct from other environments and uses a special subsystem, Predict Application Audit (PAA), to control it.

A special problem is posed when an application is implemented in multiple production environments and a change to the application is required in one but not all of them, or if the change is required in all of them simultaneously. Both of these problems are addressed with PAA in the first instance, it is imperative that each be controlled separately. Thus, a separate PAA subsystem controls each environment. In the second case the application has multiple defined locations controlled by a single PAA.

The critical nature of the production environment increases the importance of timely implementation. You may need to introduce objects into production at a specific date and time. PAA allows you to migrate an application or object to the environment ahead of time and specify the date and time that it can become active in production.

Sometimes, when a problem is detected with a production application, it is necessary to restore the previous version. When production objects cause system problems, PAA automates the process of removing them and restoring the previous versions.

Auditing and reporting are particularly important in the production environment. You may need quick answers to questions like the following:

- Which objects were active at a certain date and time (for example, the end of the month, or the point where a problem occurred)?
- Which objects are currently in production?
- What has changed in the production environment since last night?

Migrating Objects to the Production Environment

A PAA-controlled production environment corresponds to a production status defined in PAC. Thus, a PAC migration to the production status is a migration to the PAA-controlled environment.

A PAA load is the process of storing objects in the PAA-controlled environment. The load is usually executed automatically when a PAC migration event to the production status is processed; when the PAA-controlled environment is remote to PAC, the load is executed manually using the MIG-LOAD utility.

PAA jobs are different from PAC jobs. A PAA job (or load job) identifies the destination library and the objects to be loaded. The job is created during the processing of the PAC migration event. The PAA job retains attributes of the PAC migration event (for example, the Schedule Date).

Each PAA job identifies a specific load. Thus, unlike a PAC migration event, which can be run multiple times. Each PAA job must be unique. PAA uses the job number to define a unique job.

Migration Steps

The list below describes the steps in a direct migration (where the PAC and PAA systems communicate) to a PAA-controlled environment:

1. A user submits a PAC migration event to migrate objects from a test status to the production status.
2. PAC locks the object versions specified in the object list.
3. PAC reads the object versions and invokes PAA
4. PAA creates a unique job to identify the load.
5. PAA verifies that the user is authorized to submit the load.
6. PAA verifies the application, destination library, and Predict file and locks the library.
7. PAA backs up each object version that is being superseded and stores the new version in the library. PAA audits these actions and stores the audit data in the PAA system file.
8. When the load is complete, PAA unlocks the library.
9. PAC unlocks the objects that were locked in Step 2.

You can view information about the PAA load job, PAC migration event, application, library, and loaded objects.

The PAA administrator can review the job table at any time to determine the state of the PAA jobs.

Control and Production Versions of Objects

Whenever an object is migrated to a PAA-controlled production environment, PAA assigns the object a production version number. PAA stores both the production version number and the PAC version number (called the Control version) in a control record for the object.

For example, when Control Version 1 of a program is migrated from a PAC test status to production, PAA designates it Production Version 1. Assume that Control Version 2 is never migrated to production; when Control Version 3 is migrated to production, it is designated Production Version 2.

Superseded Object Versions and Backups

An object version is superseded when a load replaces it with another version. PAA automatically backs up object versions that will be superseded during a load and assigns the job number to the backup.

Backouts and Reactivated Versions

When a load abends or a problem is detected with object versions that were loaded, you can back out the load job. Backing out a job reverses all actions completed by the load: object versions that were added are purged, and object versions that were superseded are reactivated.

When a version is reactivated, it is restored from the backup and assigned a new production version number. The assignment of production versions is illustrated in the table below:

Control	Production	Remarks
1	1	Superseded by Production Version 2.
2	?	Never migrated to production.
3	2	Load job subsequently backed out.
1	3	Reactivated from backup of Production Version 1.

Finalizing a job prevents it from being backed out. When a job is finalized, the backups of superseded versions are purged. Backups are removed only when a job is backed out (which restores the backups) or finalized.

Loading a Job for Activation in the Future

Unlike other PAC migrations, a migration event to a PAA-controlled production status can be authorized and executed before the Schedule Date. When it processes a scheduled job, PAA loads the objects into the user system file / dataset for foreign objects that constitutes the production environment. However, it does not load them into the application library / dataset; they are inaccessible to the application. PAA designates the load and each loaded object as a Schedule until the PAA administrator activates the job.

When the job is activated, PAA backs up versions that will be superseded and loads the scheduled objects into the application library / dataset.

The dates and times in the following table are used to describe load jobs and the loaded objects. When a job is loaded as a Schedule, these dates and times might all be different.

Date	Description
Schedule Date	The earliest date and time the job can be activated. All PAA jobs have a Schedule Date; except for jobs loaded as Schedules, the Schedule Date is the same as the Load Date and Effective Date.
Authorize Date	The date and time the PAC migration event is authorized.
Load Date	The date and time the objects are loaded into the PAA-controlled environment.
Effective Date	The date and time the loaded objects become active in production. Except for Schedules, the Effective Date is the same as the Load Date.
Backout Date	The date and time the job is backed out.

For example, a job is scheduled for 1999-11-07 (November 7, 1999) at 18:00:00. On November 5, it is authorized at 12:02:33 and loaded into the production environment at 12:10:05.

On November 7, the PAA administrator activates the job at 18:30:30. Two hours later, after finding that an object loaded by the job is causing system problems, the administrator backs out the job.

PAA reports the following dates and times:

Schedule Date	1999-11-07 at 18:00:00
Authorize Date	1999-11-05 at 12:02:33
Load Date	1999-11-05 at 12:10:05
Effective Date	1999-11-07 at 18:30:30
Backout Date	1999-11-07 at 20:35:22

Production Applications and Libraries / Datasets

When a PAC application is migrated to production, PAA derives most of its information about the application from PAC definitions. However, the PAA application definition controls file assignments for application libraries, the Predict file where views/DDMs are stored and the location of the foreign object dataset.

Multiple production locations can be defined for a deployment (application).

To provide flexibility in mixed environments, applications in a PAA-controlled location can be populated with objects either from PAC or from outside the PAC-controlled environment. However, to protect the integrity of an application, you cannot replace a PAC object with a non-PAC object.

Deployment and Location Definitions

The PAA deployment definition consists of the following elements:

- The name of the application;
- The name of the production status;
- The Xref option (whether to include Predict Xref data in loads for the application);
- The default user system file that PAA assigns to the application's libraries when the user does not specify a file assignment in the library definition;
- The Predict file where views/DDMs and Xref data are loaded;
- The library definition for each PAA library the application uses.

Its name and the database and file numbers of the user system file in which it is located defines a library. Two libraries can have the same name if they are located in different system files. A foreign location is defined by specifying the ESY node, type, format, dataset name and volume.

You must define a deployment (application and library) and location to PAA before you can load objects for the application into the PAA-controlled environment.

Maintaining the Integrity of Deployments and Locations

PAA protects production deployments by securing file assignments and by locking an application location when objects are loaded into the library.

Once you have loaded objects into a deployment location, the file assignment cannot be changed unless the PAA administrator refreshes the deployment, which removes all objects and audit data and resets the library to a Dormant (never used) state.

Once objects or data for the application have been stored in the Predict file, the database and file numbers cannot be changed. During the load process, PAA locks the location accessed by the load.

The lock prevents another migration from concurrently loading objects into the location. When the load is complete, the lock is released.

PAA Administrator Functions

The PAA administrator can perform the online functions listed below. Some functions, such as refreshing a deployment or finalizing a job, can also be performed in batch.

Deployment and Location Functions

- Define a deployment and location to PAA.
- Modify parts of the deployment definitions.
- Refresh or purge deployments and locations. Purging a deployment or location removes all objects, PAA audit data, and the deployment or location definition. Refreshing a deployment or location removes objects and PAA audit data but leaves the definition intact.

Job Functions

- Activate scheduled jobs.
- Back out loaded jobs.
- Finalize loaded jobs.
- Purge backed-out or finalized jobs.
- Purge superseded object versions from the Natural buffer pool.

System Functions

- Activate or deactivate PAA system applymods.
- Activate or deactivate PAA user exits.
- Establish system defaults.

PAA Reports

PAA online reports provide information about deployments, locations, load jobs, object versions, and object backups.

Deployment Reports

Deployment reports include the following information:

- The load state of the deployment (active, pending, etc.);
- The date and time of the last migration for the deployment into the PAA-controlled environment;
- The deployments PAA libraries and their locations;
- The most recent jobs processed for the deployment.

Location Reports

Location reports include the following information:

- The physical location of the library;
- The deployment using it;
- Its load state;
- The date and time of the last migration into it.

Job Reports

Job reports provide the following information about load jobs:

- The application for which the migration loaded objects;
- The processing state of the job (backed out, loaded, pending, etc.);
- The user ID of the user who submitted the job;
- The PAC status from which the objects were migrated;
- The name of the PAC migration event and the date and time it was authorized;
- The user who authorized the migration event;
- The date the job became effective in production;
- The name, version, and type of each object included in the job;
- The name and version of the file translation table used in the job.

Audit Reports About Production Objects

In screens and pop-up windows, audit reports provide a wide range of information about object versions in production:

General Information

PAA reports the following information about each object version:

- The state of the object version (current, backed out, scheduled, etc.);
- The dates the object was unloaded from the PAC-controlled environment, loaded into the PAA-controlled environment, and activated in production;
- The date the object version was superseded or backed out;
- The corresponding production and Control version numbers;
- The application to which the object belongs and the library and status where it resides;
- The PAA load job that included the object version;
- The related PAC migration event.

Directory Information

PAA reports the following information from Natural directories:

- The library in which the object was saved or cataloged;
- The user who saved or cataloged the object;
- The date and time the object was saved or cataloged;
- The sizes of the saved and cataloged object;
- The Natural version, operating system, and TP system used to create, save, or catalog the object.

Summary Reports

On a single screen, summary reports provide the following information about object versions:

- Production and Control version numbers;
- Number, date, and time of the load job;
- The date the object version became effective in production;
- The earliest date the object can be activated (for Schedules only);
- The date the object version was backed out (if applicable).

User Reports

PAA provides APIs for reporting purposes. These APIs enable you to design and customize your own output reports based on PAA held data.

Control Information

The PAA administrator can display a snapshot of the current state of the PAA-controlled production environment:

- The date and time of the last migration into the environment;
- The numbers of active and pending load jobs (these jobs have locked PAA locations);
- The numbers of load jobs that have been backed out;
- The database and file numbers of the PAA system file where audit data is stored;
- The number that will be assigned to the next load job.

7 Application Control Issues and PAC/PAA Functions

This document summarizes the PAC/PAA functions that address the control issues discussed in section **Application Control**.

Inventory Management

Where is the source code for an object?	PAC stores all source code in the protected ACF system file.
Do the source and executable code match?	PAC automatically synchronizes the source and executable code for each object version.
What other objects do an object use?	PAC lists all Natural and Predict objects that a specified object uses.
Which version of an object does a calling program use?	PAC lists the name, object type, and version number of each object used by the calling program.
What is the current version of an object?	You can view the version history for an object. PAA lists the current versions of objects in a production status.
How are obsolete objects handled?	You can use a PAC archive facility to unload objects to storage media, purge them from the PAC-controlled environment, and restore them if necessary. From a production status, only superseded objects can be archived. With or without first archiving them, you can purge objects from a status or from the PAC-controlled environment by migrating them to a retire status.

Configuration Management

How are the operating environments defined?	In PAC, you specify the location of the Natural library, Predict file and foreign object dataset for each status (phase) of an application life-cycle. You can define multiple environments for each status type except Control, Archive and Retire.
What environments are defined for an application?	You can view all statuses to which a specified application is linked.
What is the location of a specific environment?	You can view the system file assignments for the linked status.
What is the life-cycle of a specific application?	The life-cycle is the structure of linked statuses for an application, along with the valid migration paths among them. You can view all linked statuses and migration paths for a specified application.
Who can authorize migrations between two environments?	Valid authorizers are specified in the migration-path definition. You can view the valid authorizers for the migration path.

Migration Management

How do you handle migrations when the origin and destination environments are heterogeneous?	PAC operates independently of the operating system, TP system, and DBMS (PAC supports ADABAS, DB2, and VSAM files).
How do you enforce migration and promotion procedures?	Standard migration and promotion procedures are defined and enforced in the migration paths, migration events, and jobs. The PAC administrator controls who can define migration paths; migration paths specify authorizers; and no unauthorized migrations can be made.
How do you ensure that the correct object version is implemented and referenced?	By default, PAC migrates the latest object version in the origin library. To ensure that the latest version is the correct one, you can display the source code for the version or compare versions.
How do you compile an object referencing one set of databases and files and execute it referencing a different set?	Assign a file translation table to the application status link for each status in which you want to execute the object. When you migrate the object, PAC dynamically recompiles it to reference the correct database and file numbers.
How can you recover an object that is accidentally overlaid?	PAC stores each object version in the protected ACF system file. If an object in a status library is overlaid, it can be reloaded. PAA backs up a production object before replacing it; you can back out the job that replaced it and restore the earlier version.

Production Management

How is implementation controlled?	All PAC migrations require authorization. The PAA administrator can predefine production applications and control load jobs.
Where is an object implemented?	PAA displays the production library, database number, file number and dataset for any object in a PAA-controlled production environment.
How is implementation of production objects tracked?	PAA automatically tracks migrations of objects into production. PAA provides audit reports and supports user-written reports via use of the supplied APIs.
How do you recover when an implemented object causes system problems?	The PAA administrator can backout a load job and automatically restores an object version replaced by the load. You can also reload earlier versions.

Change Management

How is maintenance tracked and controlled?	To change an object, you must make an authorized migration to a development or maintenance status. PAC logs all check-outs and check-ins (migrations to and from a maintenance status), which can be audited and disallowed. Maintenance requests can be linked to migration events and logs.
Who is changing an object?	The change control log for the object shows all users who have checked out the specified object.
How are concurrent changes handled?	By default, PAC warns a user who checks out an object that is currently checked out by another user; concurrent check-outs can be blocked. When several users check out an object version concurrently, each check-in of the object is given a new version number.
How are related problems grouped?	You can define a single maintenance request for related problems and assign it to multiple migration events; thus, you can view all migrations to and from maintenance for the related problems. Maintenance requests can span applications.

8 Using External Version Control with PAC

If you are using an external version control software such as Subversion (SVN) or CVS for versioning on an Open Systems platform you may want to use it also to version your Natural sources. You can already do so using the Local Versioning plug-in of Natural Studio (Natural for Windows) or the versioning functions of Natural for Eclipse or NaturalONE. However, in addtion to the external version control software PAC offers the support of the complete product lifycycle of an application from development to production environments.

Natural sources versioned with external version control software can be introduced into PAC without the need to create new versions of the sources in PACs ACF. Instead, a relationship to the versioned source text is kept.

General Architecture

Source Access and Handling

The access to a source text in an external version control software is realized by means of a `REQUEST DOCUMENT` statement using the URL of the source code file. For details on the `REQUEST DOCUMENT` statement please refer to the corresponding section in the *Natural Statements* documentation.

The link of a PAC application to the status type External defines a foreign source from where Natural sources can be transferred into a PAC controlled environment and holds the information about the path into the external version control software. Additionally, a user ID and a password may be defined to such an application-status link. User ID and password are used when accessing the source in the external version control software.

It is possible to change the versioning location of existing applications from PAC versioning to external versioning and vice versa.

Usually, applications versioned in an external version control software do not have a link to a maintenance status, because bug fixing will be done in a development environment and the changed sources are committed to the external version control software and then newly transferred into PAC.

When new or changed Natural sources are transferred into a PAC controlled environment, a transfer event from External to Control status (or any other target status) must be created. This can be done by specifying the fully qualified URL. When the transfer event is executed, the handling is the same as with other migrations from a development status into a PAC controlled environment. PAC gets the source by means of a `REQUEST DOCUMENT` statement and writes it into the PCF, compiles the sources in the PCF and creates a new version of the cataloged object in the ACF. DDMs are written into the PCF. But instead of the source text the used URL is stored in the ACF.

Before storing the source into the PCF, the line numbers and line number references will be recreated.

The Natural member name, the programming mode, the save date, the user and the code page information will be extracted from the source text. This information is either inserted into the source text by the Local Versioning plug-in or by Natural for Eclipse. If that information does not exist, default values will be used. The member type of the source file can only be derived from its extension. Those URLs that do not allow to determine the member type will be rejected.

Deployment and Maintenance

From this point on the already existing deployment mechanism of PAC can be used. If the source is accessed again, the stored URL has to be used. The function Display Object Version displays the URL of an object. If there is a need to maintain a program (e. g. for bug fixing), the URL of the object that is to be changed in PAC has to be picked. Then the library contents of the branch used when transferring the objects must be checked out.

After the changes are made and commited to the external version control software, a new branch must be created. An example of how to proceed with Subversion as an external version control software is described in the Local Versioning section of the *Natural Studio Extensions* documentation.

URL Handling

When using e. g. SVN as an external version control software, the command `svn diff <new-branch> <old-branch> --summarize` produces a list of URLs of those sources that were modified and added. This information can also be written to a text file. If workfile 7 is defined as the Entire Connection workfile such a text file can be used as input for the IMPORT command in the Migration List editor.

Caution: Please note that the above mentioned command produces only URLs that denote the head revision (the latest revision) of a source. However, since the URLs are required for future access, you are strongly recommended to use unique URLs (e. g. containing information about a specific revision, branch or both). Developers must create a branch in the external version control software before transferring the objects into PAC to make sure that no changes are committed into this new branch. The existing branch must be freezed for editing.

Alternatively, the commands ADD EVENT and GENLIST EVENT allow to import lists of URLs (denoting the objects to be transferred into PAC) either instream (option I of GENTYPE) or by means of a workfile (option R of GENTYPE) in batch. The local GENLIST command accepts a base URL or the input from an Entire Connection workfile.

Another alternative is to use the SEL command in the object list editor to produce the object list from a base URL.

Further Aspects

For Migration Paths having an External status as origin, no expansion is possible, because

- there are no Xref data available in this status and

- the contents of a directory can not be determined automatically.

Handling of Error Messages and DDMs

Natural error messages are versioned externally as complete error message files (one per language). Whenever the URL of such an error message file is specified in the migration list then a new version in PAC is created for each message contained in the file. It is not possible to specify a single error message in the migration list of an event from status External.

The source of DDMs and Error messages is stored additionally in the ACF and thus handled as in previous PAC versions once under PAC control. This is done for performance reasons.