

Configuring a PAC Environment

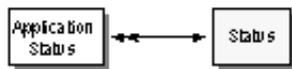
Once you have decided on the life-cycle stages for your application, you need to define the statuses you will use, you need to define your application to PAC, and you need to link the application and the statuses. You need to decide where the user data will reside for each status and for Natural objects, you may need to use file translation tables.

When you have defined the statuses and the application to PAC, and have defined the application status links between them, you need to define the migration paths between statuses that the application will use.

This chapter covers the following topics:

- Defining PAC Statuses
 - Linking Statuses and Applications
 - File Translation Tables
 - Defining Migration Paths
 - Jobs
-

Defining PAC Statuses



Each status represents a distinct phase, stage, or milestone in the life-cycle of an application. Within PAC, each status is an operational environment for an application or object. The type of status determines how PAC handles objects in that status.

- A status defines the location of objects migrated to that status.
- An object may exist in several statuses at any one time.

Status definitions are used to set up statuses in PAC. They establish the defaults to be used when the status is linked to an application. These definitions identify:

- the status type;
- the Natural system file where the Natural objects are located;
- where applicable, the associated Predict file where Predict objects such as cross-reference data, files and verifications are kept;
- options for maintaining cross-reference data for the status.

Status Types

A PAC status type (deployment) determines the rules of migration to/from the status. Although the PAC controlled environment consists of the same basic environments that are typically found in an application environment (development, maintenance, test, production), the PAC status types that correspond to these environments have differing behaviors; therefore, you should select the environment accordingly.

Using PAC, you are free to assign a status type maintenance to an application development environment or a status type test to a production environment. The status type defines the rules; you are free to use the rules as appropriate in your particular PAC system.

A status type cannot be modified; however, a status may be purged if it is not populated with objects. For instance, once an application has passed through a certain status, it may never need to pass through that status again. In those cases, the definition for that status could be removed from the application life-cycle.

All or any combination of status types may be used when establishing a user environment or identifying it to PAC. When you define a status to PAC, you must assign one of the predefined status types to it with the following exceptions:

- You may not assign the reserved PAC status types of Archive, Control and Retire; only PAC designates Archive, Control and Retire statuses.
- A corresponding history status type is defined whenever a production status type is defined to PAC. A history status is optional, however, and may be omitted.

Reserved PAC Status Types

The Archive, Control and Retire status types are reserved to PAC. There is one and only one instance of each type; that instance is predefined in the PAC system and cannot be deleted.

Archive

The Archive status type is a single status named "archive".

It is the status for any versioned object that has been removed either temporarily or permanently from the PAC environment. Objects that exist in test, production, and maintenance statuses are normally not eligible for archiving (but see Applymod 23). Active production objects cannot be archived.

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.

Control

Control is a single status named "control".

The PAC controlled environment is synonymous with the PAC reserved status "Control".

All objects are automatically placed into Control status when they first enter the PAC controlled environment, even if they are not explicitly migrated to Control. This means that the source and executable code for all versions of all objects for every PAC-controlled application are always accessible in the Control status and are centrally controlled. When reviewing objects in Control status, you can see all other statuses in which the object exists as well.

The Control status is implicitly used whenever application objects are migrated into and out of PAC; the objects need not be migrated directly to the Control status. If the destination status is a status other than Control, PAC then copies the executable object from Control to the destination library. Thus, an object in a production status could be copied to a maintenance status for changes while the original remains available to users.

If explicitly identified, the Control status may be used as a staging area and a way to distribute authority. Parts of an application may be completed and migrated into the PAC Control status, one part at a time. Once the whole application is ready for testing, it can be migrated as a unit from the Control status to statuses for which migration paths have been specifically defined.

For example, several programmers may be responsible for different parts of an application. The programmers could be allowed to 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 from Control to a

designated test status.

Although all versions of objects are accessible from Control, PAC assumes the most recent version is intended when an object is referenced by the status Control.

Retire

A retire status defines a logical location for objects to be purged

- from PAC Control or Archive.

Objects that exist only in Control status, or objects that have been removed from all other statuses and have been archived, may be physically and permanently removed from PAC, including all audit information stored about the objects.

- from another PAC status.

Retire from a status other than Control or Archive results in the permanent removal of the object(s) from the specified origin status, and the removal of the object status definition in PAC, but no deletion of the object(s) in PAC itself. Inactive or obsolete objects are good candidates for migration to a retire status. Applymod 23 must be active to retire objects from statuses other than Control or Archive.

User-Definable Status Types

Other status types can have multiple instances defined and named by the user. This is especially useful in distributed or decentralized environments. For instance, if your application is to be moved into production in more than one environment, a production status is recommended for each environment. Each user-defined status must be assigned a status type.

Development

The development environment is usually the location where all new and major ongoing application development occurs. Objects in the development environment are not part of the PAC controlled environment and source code can be modified freely.

Because the development environment has minimal controls or restrictions applied to it, needed controls or security restrictions must be implemented through Natural Security, Adabas security, or other facilities external to PAC.

A development status type defines the location of objects being developed, before they are migrated to the PAC controlled environment.

Only saved objects are migrated into the PAC controlled environment from a development status; they are cataloged in PAC as part of the migration process unless they are identified as dynamic source. Natural command processors are special cases in that they may be migrated into PAC in catalog form without being recompiled during the migration. Foreign datasets may be migrated into PAC either as source or compiled code and are not compiled by default during migration.

Maintenance

A maintenance status defines the location of maintenance libraries where saved objects that have been checked out of the PAC controlled environment can be modified to correct problems or add functionality.

Objects in a maintenance status differ from objects in a development status in that maintenance objects are always implicitly tracked by PAC. PAC automatically creates a change control log and performs check-out/ check-in controls for all objects migrated to a maintenance status and tracks their progress. This control allows PAC to keep track of and differentiate modifications made concurrently in two places; development and maintenance.

When these objects are associated with maintenance requests, PAC allows you to view all objects updated/modified for a particular maintenance request.

The maintenance environment can optionally support a test environment where changes can be tested before being returned to the PAC controlled environment.

When modified objects are returned to PAC control, the saved object is migrated into the Control status and cataloged; that is, "checked in" (unless it is dynamic source). A new version number is assigned to the object, and the old version number is kept as a reference note on the new version.

Both Development and Maintenance

Both a development and a maintenance status type are available in PAC to accommodate the requirements of applications undergoing routine maintenance and development at the same time; an essential consideration when an application continues to be developed after it is implemented in production.

Separating these two environments precludes the possibility of objects being recompiled with the incorrect subordinates; user views, rules, global data areas, maps, and copycode.

If both a maintenance status and a development status are established:

- the development status is generally used for developing the next major implementation of an application as a whole; whereas
- the maintenance status is used for required maintenance to the implementation of the application currently running in production.

If both development and maintenance statuses are defined, they need to be associated with different libraries (and Predict system files, where necessary).

Incorporate

The incorporate status defines the location of an existing application to be "taken on" when PAC is first implemented at a site. Entire applications or certain objects of an application can be incorporated into PAC. Objects incorporated into PAC will not be compiled.

This status type allows you to place under PAC control applications or objects that exist only in catalog form.

Objects can be incorporated to a Control, test, or production status type. The incorporated application establishes a base level for subsequent development, testing, implementation, and maintenance in the PAC environment.

In some cases, you may wish to incorporate your applications into PAC and never compile them in PAC, perhaps because the application originates from a remote location and should not be recompiled unless absolutely necessary.

An object may be incorporated only if a previous version of the object was not compiled in PAC. PAC does not allow an object previously migrated into PAC from a development or maintenance status type to be incorporated, because incorporation could affect the integrity of not only the object itself, but related objects and even the entire application. PAC has no way of synchronizing and controlling subordinate and other objects that may be used by the incorporated object since objects are not compiled when they are incorporated into PAC.

Test

A test status defines the location of objects in a test environment. Test statuses are used to facilitate the testing of applications before they are implemented in production. You may define several user-defined test statuses such as integration test, system test, quality assurance, and user test to accommodate the application test plan.

All objects migrated from PAC to a test status type are implicitly tracked. In principal, test status types have only execution (runtime) requirements and generally need to have objects in compiled (cataloged) form only. Objects that use dynamic source variables are an exception.

During testing, an application may be modified using SYSMAIN or the PAC migrate/load utilities. However, if PAC is not also updated with any changes made, the changes are ignored when PAC migrates objects from the test status. This restriction ensures the consistency of the application.

If you need to maintain a protected environment during application testing, you should consider establishing the test environment as a production type environment. Certain protections can also be implemented using Natural Security.

Separate libraries (and Predict system files, where necessary) are recommended for different test statuses. It is not necessary to use a separate Natural system file.

Production

The primary objective of PAC is to preserve the integrity of the production environment.

Production is normally the ultimate environment in which an application is installed for use by an organization for its business needs. Except for objects that use dynamic source variables, PAC migrates only executable code to a production status.

A production status defines the location of the PAA system files. You may define as many different production statuses as you need to accommodate a distributed production environment.

A production status type is controlled and audited by the Predict Application Audit (PAA) subsystem, an extension of PAC that enforces integrity, consistency, and control. All objects are protected by PAA and cannot be modified.

Although PAA reporting facilities are independent of the PAC reporting facilities, all objects migrated into the production environment are under PAA control and remain consistent with the versioned objects under PAC control. The status of the application and its objects in the production environment is known for any time, present or past.

PAA tracks application activities in the production environment and provides information about

- the sequence of migration events, information that may be important for resolving certain problems;
- the PAC objects that were actually migrated into the production environment;
- the objects that were backed out; and
- the objects that have been activated.

PAA also provides administrative facilities in the production environment.

Software AG recommends that the production environment be kept separate from all other PAC environments; that is, the production environment should have a separate Natural system file and a different Predict system file.

Status Locations

By default, a user-definable status points to the Predict file (DBnr/Fnr) and the Natural system files (DBnr/Fnr) that were in effect at the time PAC was initialized. You can modify these file locations to suit your site requirements.

The Predict file and Natural system file locations specified in the status definition are defaults for the status; this information can be modified for any particular link between an application and a status.

When an application is linked to a status, the physical locations of the programming objects that make up the application are specified. The Predict and Natural databases and files where all objects for the application are stored are identified.

Migration of an application or object from one status to another may or may not result in the actual physical movement of that application or object, depending on the particular application status link definition.

In distributed environments, a distributed status definition allows you to maintain multiple states of an application in parallel with one another. Such a status may be a single definition representing multiple physical environments, each of which may be located on a different database and/or machine, or it may be a number of Natural libraries on a single Natural system file.

Foreign Object Support

Non-Natural and non-Predict objects are foreign objects in PAC. Foreign object types are defined to PAC using the Foreign Maintenance Administration PAC function, allowing foreign objects to be included in the application when it is added or changed. The latter is done using the Foreign Support additional options. Once defined for an application, only those foreign objects can be assigned to related foreign locations in the application's links.

Logical Definition

A status definition is a logical definition with an associated physical location (that is, a library, database and file number).

If two different statuses for the same application have the same physical location, and a migration path is set up between them, then the migration path is considered to be a logical path because no objects are actually moved during the migration.

For example, if an application is successfully tested in the USER_TEST status and the application is passed to the ACCEPTANCE_TEST status, it has reached a new milestone in its life-cycle. If the USER_TEST and ACCEPTANCE_TEST statuses have the same physical location (that is, the same Natural library name, the same database ID, and the same file number), then any migrations between the two statuses result in no physical movement of objects; the action is a promotion of the objects to the next logical status.

Natural and Predict Files

A status may be associated with only one Predict file and one Natural system file. The Predict and Natural file locations defined in the status definition are used as the default locations.

When an application is linked to a status, you may specify a different Predict and Natural library name and physical location (database ID and file number) for that application at that status. This option is especially useful if the default status Predict file or Natural system file is not large enough to hold the information for all of the applications linked to the status.

It is recommended that the same physical Predict file be used for cross-reference data and userviews to ensure the consistency, accuracy, and integrity of the Predict information for all applications.

Special Status Location Considerations

A maintenance library can be on the same Natural system file and Predict file as the development environment, but if so, separate libraries are required to prevent ambiguities.

A test library can be on the same Natural system file and Predict file as the development environment, but if so, separate libraries should be used.

The PAA subsystem allows PAC to track applications in production. It is recommended that the production environment be separate from all other PAC environments, that is, have a separate Natural system file and Predict file.

Defining Applications to PAC



An application is a collection of entities configured to accomplish specific data processing tasks.

Most PAC activities are performed at the application level. PAC handles applications as a whole, parts of applications, and individual objects belonging to an 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.

Application Name

When objects are migrated into PAC, the migration must be assigned an application name. That application name then becomes the high-level qualifier to identify the object(s) migrated. Because the application name must always be given to identify an object, it is implied that the object is owned by the application. In this way, objects with the same name, type, and version that belong to different applications can be identified as different objects.

An application definition requires a unique name within PAC. You must then set up the default Natural library, foreign datasets, and Predict file for each of the statuses that will be part of the application test plan in your environment.

The application names Predict and Predict Case are reserved for PAC and may not be deleted. These are special-use applications.

Predict Applications

PAC uses a special application type (PRD) to accommodate data dictionary objects. A default application 'Predict' is delivered as standard and this application may not be deleted. The (PRD) type application is handled like any other application under PAC control in that it can be linked to statuses, and the Predict data can be migrated from one status to another using migration events.

The (PRD) type application is used to populate PAC with the data dictionary information from Predict. This information is used by Natural applications defined to PAC. (PRD) type application objects must be migrated into PAC before the application that uses them is populated with its respective objects.

Objects migrated for application type (PRD) can either be a shared resource or an individual one. Many Natural applications can share a single (PRD) application or even use their own individual (PRD) application.

Because Predict objects have properties that differ from those of Natural objects, PAC handles Predict objects differently. See the section Migrating Predict Objects.

Predict Case Application

The Predict Case application is used to populate the Predict Case objects used for composing objects for other applications under the control of PAC.

Predict Case application objects are automatically migrated into PAC when the Natural objects that use them are migrated into PAC.

Unlike other applications, the Predict Case application may not be linked to statuses other than Control status.

Copying an Application

Copying an application to create a new application saves time. You can create an application model appropriate for your site and use it as a template for creating additional applications.

When you copy an application, you can copy application defaults, application status links, and migration paths at the same time. Once the application is copied, appropriate default libraries can be specified in the fields provided. These library specifications will be automatically substituted in the application status link definitions for the new application.

Original Application Library

For Natural Objects

When loaded into PAC, the Natural objects of an application are grouped together as a single library. PAC currently limits an application to one library definition per status.

Outside of PAC, a library and an application may be defined as one and the same, or an application may be implemented across libraries.

Suppose an application outside of PAC is implemented across two libraries and there is a MENU program in each library. In order to define the application to PAC, it would be necessary to either group the objects from the two libraries into a single library and rename one of the MENU programs to avoid ambiguity within the application, or define each library as a separate application under PAC.

The original Natural library from which objects are loaded into PAC may be arbitrarily assigned. The execution Natural libraries where PAC delivers objects for actual execution (production or test) may be different, and the Natural library for test may be different from the Natural library for production.

Although an application may have a one-to-one or a one-to-many relationship with a Natural library, PAC currently supports only the one-to-one relationship; PAA, however, supports the one-to-many relationship.

For Foreign Objects

When loaded into PAC, the foreign objects of an application may be located in more than one dataset. Each dataset contains objects of a specific class and type (for example, COBOL, COBOL Load, Assembler Copycode). The classes and types of foreign objects supported are defined for the application.

Foreign objects (datasets) are defined for an application when the application is defined to PAC or modified. This definition limits the support for these objects later in the application status links and also migration events.

Using Other Applications as Steplibs

Another application may own or have defined to it additional and/or common subordinate objects (copycode, maps, data areas) that are required for the successful compilation of objects belonging to your application. In PAC, you can define the "library" of that other application as a step library for use at compile time.

PAC also supports steplibs for use at execution time. In certain circumstances, the Natural objects of an application in a particular library may need to invoke or use at execution time certain objects (global data areas, subroutines, programs, maps, help routines, subprograms) contained in other libraries that were populated by the same application or by other applications. Natural Security must be used to establish execution time steplibs for these libraries as the role of PAC is only to distribute the objects in a controlled and coordinated manner.

Steplibs have the following advantages:

- Duplicate or redundant objects are removed across applications.
- "Common code" is consistent. For example, when compiled, all objects for all applications use the same version of the same copycode.
- When the common code is changed, you have the option of using the Generate and Expand facilities to identify all objects across applications that were compiled against the version of the code. For example, if a piece of copycode used with the steplib facility is changed, all objects (for example, programs, subprograms, subroutines) that use that copycode may be listed for all applications.
- The rules for using an application as a steplib are the same as those for using a library under Natural.

A compile time steplib is used only when objects are compiled during migrations into PAC from development or maintenance statuses or during alignment (a Control-to-Control) migration.

Defining an Application as a Steplib

Any application defined to PAC may be established as a compile steplib for any other application. The compile steplib is specified in the application status link definition. If a default steplib is defined to the application, PAC uses it in the definition for a new application status link.

You can specify a steplib in the application status link only for development or maintenance statuses or for the Control status:

- For a migration from a development or maintenance status, the steplib must be specified in the application status link for the origin (From) status of the migration event.
- For an alignment, the steplib must be specified in the application status link for the Control status.

Example

Application BILLING uses copycode in the ACCOUNTS application; therefore, the development-to-Control application status link for BILLING must specify ACCOUNTS as the compile steplib.

Steplib Objects

The object types of a steplib that can be used during compilation are:

- copycode
- local, global, and parameter data areas
- maps

Using Steplib Objects to Compile

During the migration of objects from development or maintenance status types:

- PAC ensures that the objects in the steplib are available during compile time for any application that uses them. These subordinate steplib objects cannot be specified in the object list for the application that uses them because they do not belong to that application.
- Objects that were originally part of one application may be switched to another application. If, however, the object is a subordinate (for example, copycode), it must be removed from the PCF system file by the PAC administrator or purged from the original application. If subordinates are not found in the current application, the steplib is checked. If the object cannot be found in either location, a compile error will result.

Resulting Steplib Object Information

When the compile is performed, PAC builds versioned cross-reference information about all objects compiled. Cross-reference information includes the following additional information for objects from a referenced steplib:

- The use of objects in the steplib application.
- The name of the application that owns the objects.

In addition, the objects of a steplib application acquire "used by" references for all objects of all applications that used them at compile time. The "used by" reference is always the highest version of the object (rolling is not possible).

New Steplib Object Versions

An object in the steplib application is changed in the same manner as an object for any other (non-steplib) application; however, when the new version of the object is created, you may invoke User Exit 22 (if it is active in your environment) to obtain a list of all objects that used the previous version. This list may be used to verify consistency since the migration of objects may be performed for only one application at a time.

User Exit 22 may be used to print messages, or it may be used to write information to a work file. If the information is written to a work file, that work file may then be used as input to the batch ADD EVENT (GENTYPE and GENLIST) option, using the "I" (read the object list from input) option to generate the object list. A new event may then be run to align the objects of another application to use the new version of the steplib object.

Linking Statuses and Applications



Each application defined to PAC must be linked to a series of statuses through which it may pass during its life-cycle. Each status must be specifically linked to the application. The collective set of statuses linked to an application define the life-cycle of the application. Objects can be migrated only to the statuses and physical locations defined to the PAC system through these application status link definitions.

Notes:

1. A status must be defined to PAC before it can be linked to an application.
2. All applications defined to PAC are linked automatically to the Control status. The Control status is used internally by PAC and its location cannot be changed by the user.

Each application status link identifies, for each application

- the location of the application's Predict data (Xref data, rules, and views/DDMs) for the specified status. The Predict file is defined for each status.
- the location (name and node) of each dataset containing foreign objects of the application for the specified status. The available datasets are defined for each application.
- the physical library name, database and file number of the Natural system file where the Natural objects of the application will reside when migrated to the specified status. This is currently limited to one library.

The link definition may also specify a file translation table to be used in migrations to the status. The section File Translation Table 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. The section Using Other Applications as Steplibs describes the use of step libraries.

Application status links do not identify the sequence of migrations for an application. Allowable migration paths between statuses must be established separately for each application before a migration can occur.

Application/Status Libraries and Datasets

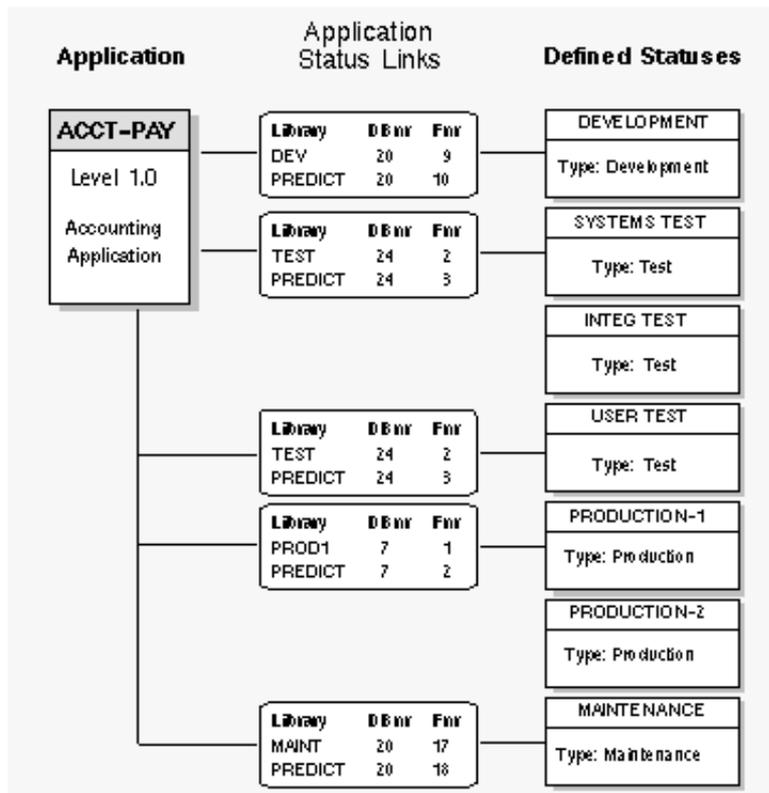
Within PAC, an application is linked to a series of statuses and a specific location is assigned to each status identified for the application. Each application has the following associated libraries and files:

- Natural libraries, each indicating the location of the Natural objects when the application is in a particular status;
- foreign datasets, each indicating the location of foreign objects of a specific class and type when the application is in a particular status; and
- Predict files, each indicating the location of the Predict data when the application is in a particular status.

The database and file number of the physical Natural library and Predict file for an application, and the node location of foreign datasets is specific to each application/status link definition, even though the status for two or more applications may be the same.

For example, suppose the applications SPARE-PARTS and GENERAL-LEDGER both have a status name of USER_TEST. However, application GENERAL-LEDGER is at physical location library GL-TEST, database number 1, file number 6, while application SPARE-PARTS is at physical location library SP-TEST, database number 2, file number 9.

The following diagram shows an application linked to a subset of the defined statuses. The figure shows how each application status link defines the Natural library, foreign dataset, 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.



File Translation Tables



The file translation table (FTT) facility is unique to PAC and its use contributes to the effective implementation of the PAC methodology. It resolves the problem of executing the database call to the correct database number and file for each application phase or environment.

When Natural objects are compiled, the views/DDMs referenced identify a set of database/file numbers of user data that the code will access/update at execution time. If an object is then migrated within PAC to an environment where the user data is located on another set of database/file numbers, PAC uses a file translation table (FTT) to translate between these database/file number sets.

When you migrate an object to a status, PAC copies the executable object from the ACF without removing it; thus neither the source code nor the executable code in the ACF is affected.

In PAC, an FTT is assigned to an application status link. During the migration process to an application status with an attached FTT, PAC dynamically recompiles the migrated copies of the objects to resolve the database/file number inconsistency without affecting any of the actual programming logic.

FTTs may be used in all Natural migrations to and from development, maintenance, test, and production status types; they can also be used in migrations from Control status to any one of the preceding status types.

FTT Versions

FTTs are versioned to ensure the integrity of the application in a migration.

The current version of an FTT can be modified if it has never been used during the processing of a migration event. The current version is used during the processing of any migration events which require that FTT.

Once the FTT version has been used in a processed migration event, it can no longer be modified. Any attempt to modify it causes PAC to create a new version and assign a new version number to it. This version supersedes any previous versions as the current version and can be modified until it has been used to process a migration event.

Previous versions remain for documentation purposes, but these versions cannot be modified.

For example, if the current version (Version 3) of an FTT has been used in a migration event and you attempt to modify it, PAC assigns Version 4 to the FTT and a "4" is displayed in the Version field on the FTT Allocation screen.

Version 4 may be modified as often as necessary until it is used in a migration event. Version 3 remains as an audit trail for any migration event that used it. The DBnr and Fnr data from the previous version are provided as the base information for the new version.

Comparison with the NTTF Macro Facility

The dynamic recompile function provided with PAC for use with FTTs offers the same functionality as the NTTF macro facility function (that is, the NATPARAM parameter for the translation facility of Natural) and in no way changes the logic of the executable code of the object. The correct database and/or file access is not dependent on the specification of special NATPARAMs or dynamic NATPARAMs.

The NTTF macro performs a function similar to that of the FTT, but with the following differences:

- The FTT makes permanent changes to the Natural objects; these changes remain in effect regardless of where the object is migrated.
- The NTTF macro assignment is dynamic, that is, a temporary solution which remains in effect only as long as it is defined in the NATPARM parameters of the current Natural session. The correct NTTF must always be specified to ensure that the correct database and/or file numbers are being accessed at execution time for each execution environment.

Why a File Translation Table?

Natural saved objects reference userviews that identify the DBnrs and Fnrs of the physical files to be accessed at execution time. When an object is cataloged, the syntax-checking facility verifies the userview and retrieves the physical file references stored in the cataloged object.

To ensure the integrity of the object and user data accessibility, the object can be executed only against the DBnrs/Fnrs identified when the object was compiled; however, when the object is migrated to an environment where the user data is located on a different set of DBnrs/Fnrs, the object must be recompiled.

Assuming that the DBID=0 option is not used, the following actions would normally be necessary to prepare and implement objects into test or production:

1. Migrate Predict userviews and files;
2. Renumber DBnrs and Fnrs;
3. Regenerate the Natural views (DDMs);
4. Recompile the Natural objects.

Alternatively:

1. Copy Natural views (DDMs);
2. Renumber the DBnrs and Fnrs using SYSDDM;
3. Recompile the necessary Natural objects.

These procedures have the following disadvantages:

- They are time-consuming.
- Inconsistencies may be introduced to the application (if, for example, something has been forgotten) requiring that the previously tested objects be retested.
- Subordinate objects (copycode, maps, data areas) used during the compile may be different from those of the environment where the code was tested, possibly introducing errors and/or inconsistencies.
- They increase the "outage" time for the actual implementation of the objects into the physical environment.

The PAC FTT facility solves the problem by dynamically recompiling ("zapping") the already compiled code with the required changes for DBnrs and Fnrs as it is migrated to the destination environment (status).

Implementing FTTs in PAC

You may use one or a series of file translation tables (FTTs) for each application. You may set up one or more base tables and then create or copy subsets for special requirements. For each table, the origin DBnr and Fnr always represent the DBnr and Fnr of the views/DDMs with which the objects were first placed under PAC's control and compiled.

If the same table is required by two different applications, you can copy it from one application to the other or you can share the table across the applications by specifying all of the other DBnr/Fnr that are valid for each application in the destination status.

Defining and Assigning FTTs

An FTT is defined using the File Translation Table maintenance facility. The table is assigned as part of the Application Status Link definition. If a table is 'assigned' before it has been 'defined', a dormant FTT is created which will not become active until the database/file translation values have been entered using the FTT maintenance facilities.

An FTT may be defined as valid for all applications and all statuses (shared). Or it may be restricted to one specific application (not shared) and/or one specific status (a separate not shared specification) to provide security or effective usage. The use of an FTT is maximally restricted when it is defined as not shared for both an application and a status; that is, the FTT can only be used when migrating a specific application to a specific status.

Even if an FTT is defined to a specific application and status, the FTT must still be defined to the application status link because there may be more than one FTT assigned to the same application and status.

Note:

The status definition is independent of the application status link definition and the FTT is an attribute of (assigned to) the application status link definition, not the status itself.

When an FTT is defined as not shared for an application only, PAC scans the application for all known databases and file numbers that its objects access. PAC then verifies that the most recent versions of views are used, and that all databases and file numbers used have been placed on the DBnr and Fnr origin list. When a migration event is created for the application, the FTT is assigned automatically, but the assignment may be overridden by the event authorizer.

If an FTT is shared (that is, it is neither application- nor status-specific), it may be either formally assigned on the application status link or dynamically assigned by the event authorizer.

Changing Definitions and Assignments

You can change an FTT assignment for a particular application link only if the FTT is defined as shared.

If the FTT is defined as not shared, its application status link assignment cannot be updated to shared. A not shared FTT can be changed to a shared FTT only by redefining the FTT using the FTT maintenance facilities, in which case, a new version of the FTT is created.

AND/OR Operand

You can set up an FTT to translate one DBnr to another DBnr; and/or one Fnr to another Fnr (the OR operand). In this case, PAC looks at and translates the DBnr and Fnr independently. Thus, an FTT set up with the OR operand could result in the following translation for each of the objects being migrated:

- Only the DBnr or only the Fnr;
- Both the DBnr and the Fnr;
- Neither the DBnr nor the Fnr.

You can also set up an FTT to translate one DBnr/Fnr combination to another DBnr/Fnr combination (the AND operand). In this case, PAC processes the database number and file number as a pair. An FTT set up with the AND operand translates any exact database number/file number pair to the corresponding exact database number/file number pair for each object being migrated. If the exact pairs are not matched, the FTT does no translation.

Verifying FTTs for a Migration Event

If the FTT is dormant, PAC assumes an error and rejects the FTT table when the migration event is processed, but continues processing normally. If no other application status links use this table, PAC purges the dormant FTT entry.

When the migration event is submitted, PAC updates the event with the current version of the FTT used during the migration. The audit facility allows the user to historically verify the specific FTT that was used in the migration.

FTT Examples

The following are examples of using FTTs.

Example 1: General

The following illustration shows excerpts from two FTTs for an application. 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
				24	

Origin Status	DBnr	Fnr	Destination Status	DBnr	Fnr
DEVELOPMENT	20	9	USER TEST	27	1
	20	10		27	2
	20	11		27	3
				27	

In Development status, the source code for the application objects references Fnr 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.

PAC then copies the executable object from the ACF and dynamically recompiles it, changing DBnr 20 and Fnr 9, 10, and 11 to DBnr 24 and Fnr 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 Fnr 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 Fnr referenced in Development to those referenced in User Test.

Example 2: Migration into PAC

Objects are migrated from DEV status to TEST1 status.

A number of objects are migrated into PAC from a development status type and compiled using three (3) views/DDMs as subordinates.

Note:

These DDMs must have been previously generated as a result of a Predict event.

The resulting compiled code accesses the following database/file numbers:

```
Employee DBnr 1, Fnr 3
Finance   DBnr 1, Fnr 4
Auto      DBnr 1, Fnr 5
```

A file translation table (FTT) is set up with the OR operand to translate databases or file numbers between the origin DBnr/Fnr of the development (DEV) status and the destination DBnr/Fnr of the TEST1 status as shown below:

Origin Status	DBnr	Fnr	Destination Status	DBnr	Fnr
DEVELOPMENT	1	3	TEST1	2	13
	1	4		2	14
	1	5		2	15

When the objects are loaded into the TEST1 environment, they are dynamically recompiled and translated as follows:

- DBnr 1 is translated to DBnr 2;
- Fnr 3 is translated to Fnr 13;
- Fnr 4 is translated to Fnr 14; and
- Fnr 5 is translated to Fnr 15.

The resulting database and file numbers that the objects will access/update will be the physical locations of the Employee, Finance, and Auto files in the TEST1 status; that is, the Employee, Finance, and Auto files are located on Database 2 and File Numbers 13, 14, and 15, respectively.

Example 3: Migration Between Test Statuses

The same objects (from Example 2) are then migrated from TEST1 status to TEST2 status.

Because objects are compiled in the Control environment using the views of the DEV status, and because compiled objects are always migrated from Control, the objects have the same origin database and file number (that is, DEV) as they did in Example 2.

The destination DBnr and Fnr is the user data of the files at the TEST2 status; that is, the Employee, Finance, and Auto files are located on Database 3 and File Numbers 23, 24, and 25, respectively.

A file translation table (FTT) is set up to translate databases or file numbers between the origin DBnr/Fnr of the development (DEV) status and the destination DBnr/Fnr of the TEST2 status as shown below:

Origin Status	DBnr	Fnr	Destination Status	DBnr	Fnr
DEVELOPMENT	1	3	TEST2	3	23
	1	4		3	24
	1	5		3	25

When the objects are migrated from TEST1 into the TEST2 environment, they are dynamically recompiled and translated as follows:

- DBnr 1 is translated to DBnr 3;
- Fnr 3 is translated to Fnr 23;
- Fnr 4 is translated to Fnr 24; and
- Fnr 5 is translated to Fnr 25.

The resulting database and file numbers that the objects will access/update will be the physical locations of the Employee, Finance, and Auto files in the TEST2 status.

Example 4: Migration into Production Status

The same objects (from Example 2) are now migrated from TEST2 status to PROD status.

Again, because objects are compiled in the Control environment using the views of the DEV status, and because compiled objects are always migrated from Control, the objects have the same origin database and file number (that is, DEV) as they did in Example 2.

The destination DBnr and Fnr is the user data of the files at the PROD status; that is, the Employee, Finance, and Auto files are located on Database 4 and File Numbers 33, 34, and 35, respectively.

A file translation table is set up to translate between DEV and PROD as shown below:

Origin Status	DBnr	Fnr	Destination Status	DBnr	Fnr
DEVELOPMENT	1	3	PROD	4	33
	1	4		4	34
	1	5		4	35

When the objects are migrated from TEST2 into the PROD environment, they are dynamically recompiled and translated as follows:

- DBnr 1 is translated to DBnr 4;
- Fnr 3 is translated to Fnr 33;
- Fnr 4 is translated to Fnr 34; and
- Fnr 5 is translated to Fnr 35.

The resulting database and file numbers that the objects will access/update will be the physical locations of the Employee, Finance, and Auto files in the PROD status.

Defining Migration Paths



The migration path is the route between two statuses that an application is permitted to follow during its planned life-cycle. Migration paths secure the restrictions on movement of objects between statuses. Every status and application status link specified in a migration path definition must already be defined to PAC. Events that initiate the migration process may be defined only for two application statuses that have been paired in a migration path.

The information specified in the migration path definition is used by migration events. The migration path definition specifies the origin status (the environment in which the objects currently exist and from which they are to be migrated) and destination status (the environment into which you wish to migrate the objects).

In addition, a migration path definition specifies defaults for the way a migration event that uses the path is to be processed, including

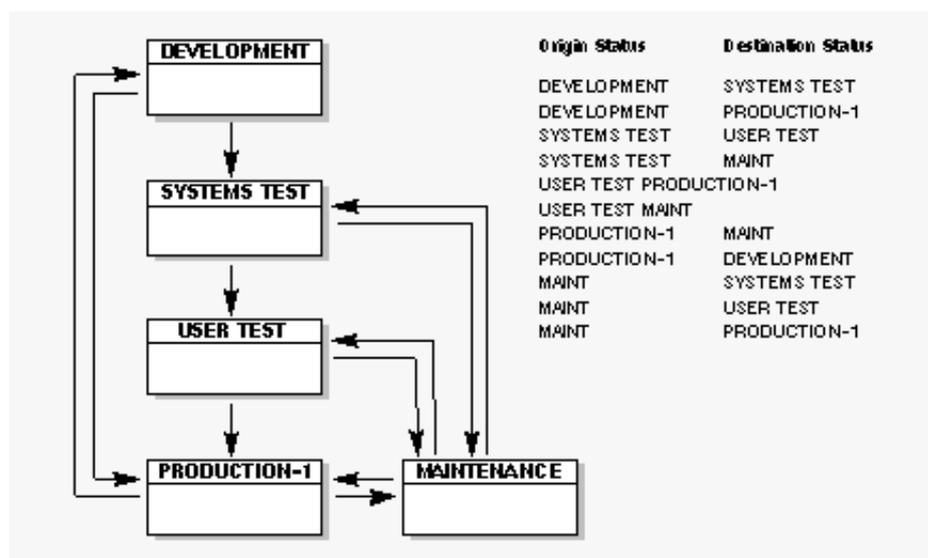
- the authorization controls required: the person(s) or group(s) who may authorize the events;
- the processing mode: whether the events will run online or in batch;
- for batch events, the names of the PAC jobs (JCL/JCS) to be submitted by the event;
- the Workfile Usage option: whether PAC will migrate objects directly from one status to another or use a work file in an intermediate step;
- the migration mode: whether the objects are to be moved or copied into the destination status when the event is processed; or, in the case of development, maintenance, and incorporation status types, whether the objects are to be included from a previously created work file.
- the Auto Expansion option: whether PAC automatically includes related objects in (expands) the list of objects to be migrated and how PAC identifies which versions of the objects to include.

It may be possible to override these defaults when the migration event is authorized. Refer to the section Customizing the Authorization Process.

Origin and Destination Statuses

The application test plan consists of the application status links and migration paths. The test plan depends on your site requirements. An application test plan may be as rigid or as flexible as needed.

Suppose that an application ACCT-PAY has been linked to each of the statuses shown in the following diagram. The arrows between the statuses indicate the paths that application objects are allowed to follow during migrations. This is the test plan for the application.



Note:

When the origin status is outside of PAC, objects are always implicitly migrated to Control status before being migrated to the specified destination status.

The allowed migration paths in the example test plan are indicated in the following table:

	Development	Systems Test	User Test	Production	Maintenance
Development				x	
Systems Test	x				x
User Test		x			x
Production	x		x		x
Maintenance		x	x	x	

Restrictions on Migration Path Statuses

- The migration path to load Predict objects into PAC must always have a destination status of Control.
- A migration path may not be defined between two development statuses, two maintenance statuses, two incorporation statuses, or any combination of these three status types.
- Incorporation is an origin status only.
- The Control-to-Control migration path is used only for the special migration type Alignment.

Event Authorizers

To ensure the integrity of the application, the decision to migrate any or all of the objects of an application to a particular status must be made at the right time by qualified people. The list of (up to eight) person(s) or group(s) who may authorize the migration events associated with a migration path is specified in the migration path definition. For this reason, the definition of migration paths is a restricted function controlled by an option on the user's profile.

If you are a person authorized (in the user profile) to define a migration path, you have the following options for specifying a migration event authorizer:

- Individual User

By specifying specific user IDs, you may individually add a person or persons as event authorizers to any migration path for any application. The user ID, however, must be defined in the user profile as being allowed to authorize migration events.

- Group

If you want the flexibility of adding new authorizers or deleting existing authorizers, you may specify the name of a group that has been defined in Natural Security. This allows you to update the authorizers for several migration paths in one place; that is, in Natural Security.

- Unrestricted or Range

The @ option implies no authorization, although the user needs to go through the step to "freeze" the event.

If authorization of the migration event is not restricted, you can use an asterisk (*) to allow any individual users who have the necessary setting in their user profiles to authorize their own or another's event. For example, @* allows any qualified user ID (not a group) to authorize the event.

You may use range notation to specify authorizers; all user IDs starting with a specific prefix, for instance. This allows new persons to be added to the list automatically. For example, @US1* allows any user whose user ID starts with US1 to authorize the event.

- Other Than Creator

If you want to ensure that someone other than the user or group member who created the event is to authorize it, you may add this restriction by using the Separate Authority Required option in the migration path definition.

Customizing the Authorization Process

The migration event authorization process can be customized using PAC system applymods, user exits, and APIs.

For instance, a migration event authorizer may be allowed to override default values specified in the migration path: whether the event is processed in batch or online, job name, or whether objects are to be copied or moved, or included from a work file.

Applymods

Applymods are activated by the PAC administrator and are used to protect the application during migration by enforcing values for the authorization process. For example, applymod 1 ensures that PAC always uses the default values supplied in the migration path by disallowing any modification of those values during event processing.

User Exits

The PAC administrator can set up user exits to increase flexibility during the migration authorization process. For example, the authorization exit, User Exit 30 (PACEX030), can be used during the authorization process

- to automatically override the default parameters specified on the actual migration path;
- to provide additional audit and security checking of the authorization process;
- to suppress display of the authorization screen.

Application Program Interface (APIs)

The PAC Express APIs provide a useful base for creating a custom system to process migration events quickly and easily. It allows you to change the way events are authorized. For example, the screen displayed can be simplified, or a PF key can be assigned to confirm an authorization.

The Event Authorization API allows you to add or purge information about the authorization for a particular migration event. This API also allows you to display authorization information.

Migration Mode : Copy / Move / Include

The migration mode is set by specifying whether the objects are to be copied, moved, or included from a work file into PAC or between statuses.

Copy Option

Copy maintains the object version in the origin status and stores a copy in the destination status; you can specify the Copy option for Natural objects, foreign objects, and Predict objects.

Move Option

Move removes the object version from the origin status and stores it in the destination status.

If the location of the origin status is

- local, the object version is loaded to the destination and the objects are purged automatically as part of the migration process at the origin location.

- remote (that is, PAC cannot issue a database call to access/update the objects), objects must be purged at the origin status using other methods such as the PACEXDEL user exit.

PACEXDEL allows you to issue delete commands through the SYSMAIN utility or the PAC migration utility MIGUNLD. Multiple delete requests may be issued each time the utility is invoked.

Note:

Predict objects may not be moved.

Include Option

The include option allows you to include in a migration objects located in a previously created work file as opposed to migrating objects directly from the Natural system file.

Include copies objects from an external work file rather than directly from a Natural system file and stores them in the destination status. The include option can only be specified for Natural objects and is valid only for batch migrations into the PAC-controlled environment from development, maintenance, or incorporation statuses.

Processing Mode : Batch or Online

A migration event may be processed either online or in batch.

- An event processed online is processed in real time in your current Natural session and the results may be viewed interactively as the processing takes place.
- An event processed in batch is processed as a background task. This is useful for large events or when processing should be asynchronous to the user's current activities.

Auto Expansion Option

The migration path definition specifies the type of object expansion that is to occur during migration event processing. A status may be specified as the Expand status so that subordinate objects added to the list during the expand process are

- migrated to a destination status from the specified Expand status.
- rolled in from the Expand status in the case of migration from a development or maintenance status.

As an option in the migration path definition, the Expand option ensures that the subcomponents to be included in the object list are determined based on the overall PAC environment at the time the migration event is processed.

User Exit 23 may be invoked during expand processing.

User Exits and APIs for Migration Paths

PAC user exits can be used to implement audits and security for the migration path.

APINPATH, the PAC Application Program Interface (API) for migration paths, allows you to add, modify, or display information about a particular migration path.

Contact your PAC administrator for information about user exits or APIs that may be operating in your environment.

Jobs



Generally, PAC jobs are used to migrate application objects between statuses. However, other jobs (for example, batch Natural jobs, Adabas utilities) may also be defined and submitted from PAC.

The batch option is the default for migrations and a job must be supplied for batch migrations. The job may be different for different types of migrations. Refer to the following Part II sections on migration for more information.

Working examples of PAC jobs are provided with PAC for all supported operating systems.

Because PAC jobs can be modified to accommodate the operational, restart, and recovery needs of a particular site, they serve to establish standard procedures for migrating objects between environments.

Appendix B presents detailed job information and lists the operating-system-dependent jobs provided with PAC.

Multiple-Step Jobs

Natural (and foreign) migrations are normally run as batch jobs that include two or three steps. Although they may be run as one-step jobs in some cases, such jobs are not recommended. The important difference is that normal, multiple-step migration jobs allow you to check the condition code of the previous job step to decide whether the next or any subsequent job steps should be executed; or whether the job should be terminated to resolve errors.

The sample jobs provided with PAC are all multiple-step jobs that allow you to check condition codes between each step. The reasons for using multiple-step jobs are as follows:

- If the origin and destination statuses are located at different locations or on different CPUs, the migration load step must be run in the required physical environment.
- The unlock step should be run only after the successful completion of the MIGLOAD step has been verified, especially when the migration load step is run at a different location. It is especially important to run the unlock step (TRNUNLOK) separately because it indicates that the migration is complete and, if the Move option has been specified, results in the migrated objects being purged from the origin status.
- For a migration from an incorporation or development status that specifies the Include option, an externally created work file is used as input. In this case, a multiple-step job must be used to ensure that the original input dataset is not overwritten during subsequent migration processing. Multiple-step jobs are strongly recommended whenever work files are used.

Online, Natural (and foreign) migration processing simulates the multiple-step job. If an error is detected in the online display of the audit report, you can decide to terminate the migration process to resolve the error by entering a Natural command at the MORE prompt. After each simulated step, the NEXT prompt appears, again allowing you to interrupt the migration process.

Processing Job Steps

Job steps are processed as described in the following paragraphs unless otherwise stated in this section (for example, Archiving).

The job steps required depend on whether the job is being processed

- online or in batch without the work file option, with or without the Include option;
- in batch with the work file option, with or without the Include option.

If the work file option is not selected, objects are copied directly from Control to a destination status. If the work file option is selected, objects already loaded into PAC Control, whether compiled or not, are unloaded to a work file for subsequent load to a destination status.

In order to process migrations online or without the (output) work file option, PAC must have direct access to the destination status specified on the application links in the migration path definition. This is the case if all environments defined to PAC are located in

- the same physical database;
- multiple databases running with the same SVC or ID Table Manager; or
- multiple databases connected through a network.

If none of these conditions is met, then a work file or some other intermediate medium of transport must be used for migration processing.

If the Include option is specified, the first job step (TRNEVENT) reads from a work file which is defined to PAC as Work File 1 (CMWK01).

Job Steps: Batch With Work File Option

Using Copy/Move Option

Step	Name	Process
1	TRNEVENT	The TRNEVENT job step validates the object list and, if the Expand option is used, expands it to include the specified subcomponents. If the origin status is development, incorporation, or maintenance, new object versions are created, the objects are read directly from the Natural library (or foreign dataset) specified in the application status link definition into the Control status, and are compiled. If the origin status is not development, incorporation, or maintenance, the existing specified object versions are locked for the application. New or existing object versions, as appropriate, are then promoted to the destination status, if it is other than Control, and are unloaded from Control to the dataset specified for Work File 1 (CMWKF01).
2	MIGRATE	The MIGRATE job step is invoked using the LOAD,ALL,* command. The object versions are read from Work File 1 (CMWKF01), which must specify the output dataset created in Step 1, and are loaded to the Natural system file and library specified for the destination status in the application status link definition. If the destination status is in a different or remote physical environment, the MIGRATE job step must be run as a separate job in that environment.
3	TRNUNLOK	The TRNUNLOK job step results in objects being purged from the origin status if the Move option is specified for the migration event. The application and the migrated object versions are unlocked and the migration is updated to the Completed state.

Using Include Option

Step	Name	Process
1	TRNEVENT	The object list can contain only a single entry. The entry may specify a single object, a range of objects (using range notation), or "*" to include all objects read from the work file. The TRNEVENT job step reads the supplied work file(s) and selects the objects according to the entry specified in the object list. It then validates and versions the objects included. The objects are read from Work File 1 (CMWKF01), which must have been previously created using the PAC MIGUNLD or the Natural ULDMAIN or NATUNLD utilities. If user error messages are to be processed, they are read from Work File 2 (CMWKF02). The dataset specified for Work File 2 must have been previously created using the Natural ERRULDUS utility. New or existing object versions, as appropriate, are then promoted to the destination status if it is other than Control.
2	TRNUNLD	When using the Include option, TRNUNLD is run as an additional job step separate from TRNEVENT to preclude overwriting the original input work file used in Step 1. Objects to be migrated are unloaded to Work File 1. The name specified for Work File 1 must be different from the original input dataset supplied for TRNEVENT; otherwise, that dataset will be overwritten. The dataset created in this step (TRNUNLD) must be used as input for the next step (MIGRATE).
3	MIGRATE	The MIGRATE job step is invoked using the LOAD,ALL,* command. The object versions are read from Work File 1, which must specify the output dataset created in Step 2, and are loaded to the Natural system file and library specified for the destination status in the application status link definition. If the destination status is in a different or remote physical environment, the MIGRATE job step must be run as a separate job in that environment.
4	TRNUNLOK	The TRNUNLOK job step unlocks the application and the migrated object versions and updates the migration to the Completed state.

Example Batch Job: Work File Option

If a migration event using a work file is processed in batch and the Copy or Move option is specified, the sample job supplied with PAC is MIGRATE_WORKFILE (or MIGRATE_WORKFILE for OS/390 and BS2000/OSD).

If a migration event is processed with the work file option and the Include option specified, it must be run in batch to supply the objects to be migrated. INCORPORATE is the sample job supplied with PAC.

- Depending on the origin and destination status combination, you may be able to select the Replace (Y/N) option. If the Replace option is not selectable, the required value is automatically inserted and the field is protected.
- You may specify a valid Auto Expansion option. If you do, you may optionally specify the status from which the objects should be included.
- If applymods have been set up in your environment by your PAC administrator, you may activate or deactivate them.

A batch migration that uses a work file and the Copy/Move options may be run as a single job step. The job syntax requires Work File 1 (CMWKF01) to be specified in the job and the following CMSYNIN cards:

```
LOGON SYSPAC
TRNEVENT @EVENT
MIGRATE
LOAD,ALL,*
TRNUNLOK @EVENT
```

When the Include option is specified, the job may not be run as a single job step to preclude overwriting the original input dataset.

Job Steps: Online or Batch Without Work File Option

Using Copy/Move Option

Step	Name	Process
1	TRNEVENT	The TRNEVENT job step validates and expands the object list to include the specified subcomponents if the Expand option is used. If the origin status is development, incorporation, or maintenance, new object versions are created, the objects are read directly from the Natural library (or foreign dataset) specified in the application status link definition into the Control status, and are compiled, if appropriate, or incorporated. If the origin status is not development, incorporation, or maintenance, the existing specified object versions are locked for the application. New or existing object versions, as appropriate, are then promoted to the destination status if it is other than Control. The promoted objects are then copied directly from PAC Control to the destination status specified on the application status link definition.
2	TRNUNLOK	The TRNUNLOK job step results in objects being purged from the origin status if the Move option is specified for the migration event. The application and the migrated object versions are unlocked and the migration is updated to the Completed state.

Using Include Option

It is possible to process a migration without the (output) work file option, but still use the Include option to supply the list of objects to be migrated from an externally created (input) work file. This is because the (output) work file option indicates how the objects are migrated to the destination status, not how the objects are supplied as input to the migration for processing.

The TRNEVENT and TRNUNLOK steps described in the previous section Using Copy/Move Option apply except that the objects to be migrated are read from Work File 1, not directly from the Natural library (or foreign dataset) specified on the application status link definition.

Work File 1 (CMWKF01) must be specified in the job for the TRNEVENT step and the rules for creating the object list using the Include option apply. See the section Batch Job With Work File Option, Using Include Option earlier in this section.

Example

If a migration event is processed in batch without a work file and the Copy/Move option is specified, the sample job supplied with PAC is MIGRATE.

If the Include option is used, you can copy and modify the MIGRATE job to include the information needed to define Work File 1. A coding example can be found in the definition of Work File 1 in the INCORPORATE job for the TRNEVENT job step.

- You may specify a valid Auto Expansion option. If you do so, you may optionally specify the status from which the objects are to be included.
- If set up in your environment by your PAC administrator, you may activate or deactivate applymods.

A batch migration without a work file using the Copy/Move options may be run as a single job step. The job syntax requires the following CMSYNIN cards:

```
LOGON SYSPAC  
TRNEVENT @EVENT  
TRNUNLOK @EVENT
```

User Exits for Jobs

PAC user exits can be used to

- implement tracking and customized security for jobs;
- make substitutions to the JCL/JCS before a job is submitted;
- specify default parameters that you can modify for batch job submission.

Contact your PAC administrator for information about user exits that may be operating in your environment.