COGNOS(R) APPLICATION DEVELOPMENT TOOLS

P O W E R H O U S E (R) 4 G L

FOR MPE/IX

MIGRATION PLANNING GUIDE

COGNOS®
THE NEXT LEVEL
OF PERFORMANCE™

Product Information

This document applies to PowerHouse$^{(R)}$ 4GL version 8.4 and may also apply to subsequent releases. To check for newer versions of this document, visit the Cognos support Web site (http://support.cognos.com).

Copyright

Table of Contents

About this Book

Overview

This book describes how to plan your migration of PowerHouse applications from MPE/iX to a UNIX or Windows platform. It is intended for system managers and developers who are familiar with the MPE/iX environment and have a basic knowledge of UNIX or Windows.

Unless otherwise indicated, where "Windows" appears in the documentation, it applies to the Microsoft Windows operating systems. And unless otherwise indicated, where "relational database" appears in the documentation, it refers to a relational database management system (RDBMS).

Chapter 1, "Migration Choices", gives an overview of the types of migration covered in this guide. It also identifies the factors which will influence your choices for each type of migration.

Chapter 2, "The Target Platform: UNIX", discusses differences between MPE/iX and UNIX that will have to be addressed during the migration.

Chapter 3, "The Target Platform: Windows", discusses differences between MPE/iX and Windows that will have to be addressed during the migration.

Chapter 4, "The File Systems", contains a separate section for each of the MPE/iX file systems: IMAGE, KSAM, MPE, and relational. In each section, we identify the target file systems available for the given source file system, and then discuss the work involved in migrating between the two file systems.

Chapter 5, "The Migration Plan", describes the 12-step plan to actually perform the migration. This identifies everything from identifying training needs to editing data definitions. We'll show two ways of doing this, with and without the Axiant 4GL migration tools.

Cognos PowerHouse 4GL, Axiant 4GL, and PowerHouse Web Documentation Set

PowerHouse 4GL, Axiant 4GL, and PowerHouse Web documentation includes planning and configuration advice, detailed information about statements and procedures, installation instructions, and last minute product information.

Objective	Document
Get detailed reference information for PowerHouse 4GL and PowerHouse Web	Cognos PowerHouse 4GL Reference documents on the documentation CD. They provide detailed information about PowerHouse rules and each PowerHouse component. The documents are: • *Cognos PowerHouse 4GL PowerHouse Rules* (PHRules.pdf) • *Cognos PowerHouse 4GL PDL and Utilities Reference* (PDL.pdf) • *Cognos PowerHouse 4GL QDESIGN Reference* (QD.pdf) • *Cognos PowerHouse 4GL PowerHouse and Relational Databases* (PHRDB.pdf) • *Cognos PowerHouse 4GL PHD Reference* (PHD.pdf) • *Cognos PowerHouse 4GL QUIZ Reference* (QZ.pdf) • *Cognos PowerHouse 4GL QTP Reference* (QTP.pdf) Available from the PowerHouse 4GL documentation CD or from: http://support.cognos.com and http://powerhouse.cognos.com
Get an introduction to PowerHouse 4GL	*Cognos PowerHouse 4GL Primer.* This document provides an overview of the PowerHouse language and a hands-on demonstration of how to use PowerHouse. Available from the PowerHouse 4GL documentation CD or from: http://www.cognos.com/products/powerhouse/download.html
Install PowerHouse 4GL	*Cognos PowerHouse 4GL & PowerHouse Web Getting Started* books. These documents provide step-by-step instructions on installing PowerHouse 4GL on each supported platform.

Objective	Document
Get a summary of what's new in PowerHouse 4GL	*PowerHouse 4GL New Features*. This document contains a summary of new features in this release of PowerHouse. It also contains documentation for features specific to PowerHouse for Windows Available from: http://support.cognos.com and http://powerhouse.cognos.com
Attend training courses	Instructor-led classes are available for PowerHouse 4GL, Axiant 4GL, and PowerHouse Web. They provide hands-on education about these products. Available courses include: • *PowerHouse 4GL - Introduction* • *PowerHouse 4GL - QUIZ Reporting By Users* • *PowerHouse 4GL - QUIZ Advanced Reporting* • *PowerHouse 4GL - Advanced Part I* • *PowerHouse 4GL - Advanced Part 2* • *PowerHouse 4GL - Relational Interface Part I* • *PowerHouse 4GL - Relational Interface Part II* • *PowerHouse 4GL - Moving to Relational: Understanding Relational Databases* • *PowerHouse 4GL - Moving to Relational: Porting PowerHouse Applications* • *Axiant 4GL - Building Applications* • *Migrating Applications with Axiant 4GL* • *PowerHouse Web - Developing Applications* Course descriptions and schedules are available from: http://support.cognos.com and http://powerhouse.cognos.com
Get detailed reference information on Axiant 4GL	*Axiant 4GL Help Reference Manual* is the online help for Axiant 4GL, a client/server Application Development Tool that allows you to build complex distributed business applications. Available from: http://support.cognos.com and http://powerhouse.cognos.com

Objective	Document
Get started with Axiant 4GL	*Axiant 4GL Getting Started* describes how to install Axiant 4GL. It is intended for system managers familiar with the Windows environment. Available from: http://support.cognos.com and http://powerhouse.cognos.com
Introduce yourself to the basics of Axiant 4GL	*A Guided Tour of Axiant 4GL* contains hands-on tutorials that introduce the Axiant 4GL migration process and screen customization in Axiant 4GL. Available from: http://support.cognos.com and http://powerhouse.cognos.com
Review late-breaking changes	*Cognos PowerHouse 4GL, Axiant 4GL, & PowerHouse Web Release and Install Notes*. These documents provide last minute product information or corrections to the documentation, which may be useful as you install and use PowerHouse, Axiant, and PowerHouse Web.
Start using PowerHouse Web	*Cognos PowerHouse Web Planning and Configuration book*. This document introduces PowerHouse Web, provides planning information and explains how to configure the PowerHouse Web components. **Important:** This document should be the starting point for all PowerHouse Web users. Also available from the Administrator CD or from: http://support.cognos.com and http://powerhouse.cognos.com
Install PowerHouse Web	*Cognos PowerHouse 4GL & PowerHouse Web Getting Started* books. These documents provide step-by-step instructions on installing PowerHouse Web components on each supported platform.
Get detailed information for developing PowerHouse Web applications	*Cognos PowerHouse Web Developer's Guide* provides detailed reference material for application developers. Available from the Administrator CD or from: http://support.cognos.com and http://powerhouse.cognos.com

Objective	Document
Administer PowerHouse Web	The *PowerHouse Web Administrator online Help* provides detailed reference material to help you during configuration of the PowerHouse Web Administrator.
Get a summary of what's new in PowerHouse Web	*PowerHouse Web New Features*. This document contains a summary of new functionality in this version of PowerHouse Web. Available from the Administrator CD or from: http://www.cognos.com/products/powerhouse/download.html

Supported Environments

For the current list of supported environments, see http://support.cognos.com.

Chapter 1: Migration Choices

Overview

This introductory chapter gives an overview of the types of migration covered in this guide. Depending on your application, you may be able to make your migration choices directly from this top-level chapter, or you may need to go into the detail chapters to aid in the decisions. In either case, you can use the detail chapters that follow to understand and estimate the scope of work involved in the migration.

Key Migration Decisions

The key decisions to make in planning your migration from MPE/iX are:

- ❑ What is the target platform?
  - UNIX
  - Windows
- ❑ What are the target file systems?
  - IMAGE emulators
  - relational
  - indexed
  - direct or sequential
- ❑ What is the end-user UI?
  - Terminal
  - Windows GUI (using Axiant)
  - Web browser (using PowerHouse Web)
- ❑ What is the migration method?
  - Using Axiant migration tools
  - Using editors and UNIX scripts
- ❑ What is the migration timing?
  - Should you do all of the above migrations at the same time? If not, in what order should you do them?

Choosing the Target Platform

The most common choices are UNIX and Windows. Factors affecting this decision are

- Scale of the application

  Small to medium sized applications (generally those that are for the workgroup or department) can run on either UNIX or Windows, but medium to large sized applications (departmental or enterprise) are best suited to larger UNIX environments.
- Cost vs. performance

  Hardware, software, and services may cost less in a Windows environment. However stronger UNIX machines may offer better response times.
- Platform knowledge

  Are there other applications currently running (or planned) on either platform? Is there a company or department standard? Does your IT department have experience with either platform? This also affects the time it takes to migrate.

- File system

  Have you already decided on a file system which is only present on one of the platforms? (Note that a relational database does not have to be on the same platform that is running your PowerHouse application. For example, an application on Windows could use ORACLE on UNIX.)
- End-user UI

  Have you already decided on using the Windows GUI (using Axiant)? This only runs on Windows (although the server could be on UNIX in a thin-client setup).
- Does your system have any 3GL applications or programs in use (either in separate applciations or in external subroutines)?

  Converting large 3GL applications to relational can be a huge task, while using an IMAGE emulator makes the task relatively straightforward.

Detailed information on differences in PowerHouse between MPE/iX and UNIX can be found at (p. 15).

Detailed information on differences in PowerHouse between MPE/iX and Windows can be found at (p. 27).

Choosing the Target File Systems

The file systems that you migrate to will depend on the file systems you are currently using. You would normally decide first on the file system type (IMAGE emulator vs. relational vs. indexed). You can then decide on the specific file system within the chosen type. Factors affecting this decision are

- Knowledge: Does your IT department have experience with one of the file systems? Are there other applications currently running (or planned) on one of the file systems?
- Simplicity: The migration process is simplest (and thus fastest) if you keep to the same file system type that you have in your MPE/iX application. Your dictionary and programs will require minimal changes, and your users will observe a minimum of behavior differences.

File System	Simplest Migration	Other Options
IMAGE	IMAGE emulator	Relational, Indexed
Relational	Relational	
KSAM	Indexed	IMAGE emulator, Relational
MPE (Direct, Sequential)	Direct, Sequential	
MPE (Relative)	Indexed	(Relative isn't available on UNIX or Windows)

- Cost and availability: These are important factors to consider, but are outside the scope of this guide.
- Performance: IMAGE is very fast and efficient, however the performance of an IMAGE emulator depends on the emulator itself. Indexed files may be fastest for simple control file applications. Relational databases can be even faster if you modify your application to use cursors. (Cursors process large sets of data at a time. The trade-off is that this requires application changes.)
- Functionality: Relational databases offer the greatest functionality. They have many automated features, such as query optimization and database integrity processing. They also offer many administration features to simplify maintenance.
- Scale: Relational databases are the most adept at handling large amounts of data. (Note that some IMAGE emulators use relational databases internally, thus offering some of the same advantages.)

Keep in mind that the file systems do not have to be on the same machine, or even platform, as the one your application is moving to. For example, your UNIX application could use DataDirect ODBC to access an MS SQL Server database on Windows.

Detailed information on differences between the various source and target file systems can be found at "The File Systems" (p. 39).

Choosing the End-User UI

The choices for an end-user interface are terminal, Web browser (using PowerHouse Web), and Windows GUI (using Axiant). Factors affecting this decision are

- User knowledge: a terminal interface may be the best choice to minimize change for existing users who use a terminal interface already. If you anticipate many new users, a Web or Windows interface may be more intuitive and thus simplify training.
- Developer knowledge: the developers writing applications with Axiant have to know Axiant's object-based environment. Similarly, the developers writing applications with PowerHouse Web have to know some HTML.
- Platform: a Windows GUI (graphic user interface) is only available if you are using Axiant 4GL, either in fat-client mode on Windows or in thin-client mode with the server on UNIX or Windows.
- Company needs, user needs, company standards, and cost: These are important factors to consider, but are outside the scope of this guide.

This planning guide does not cover this topic in further detail, since we do not recommend changing the end-user UI at the same time as changing the platform and file system. (See "Choosing the Migration Timing" (p. 14).) For further information on Axiant and PowerHouse Web, see "Cognos PowerHouse 4GL, Axiant 4GL, and PowerHouse Web Documentation Set" (p. 6).

Choosing the Migration Method

You can choose to use Axiant's migration tools to assist the migration, even if you are not migrating to Windows or an Axiant end-user UI. These tools include:

- a Migration Profile wizard to migrate subsets of the application
- a tool to convert between source and target file systems (for example, from IMAGE to Oracle) and identify definitions requiring change (for example, coded records)
- a tool to identify program code requiring change as a result of the platform or file system change, and another tool to make many of these changes automatically.

The alternative is to do the migration without Axiant, which can have both advantages and disadvantages. Factors affecting this decision include:

- Developer knowledge: if your developers already know Axiant, then the migration process with Axiant requires less of a learning curve.
- Application scale: with large applications, the time invested in setting up Axiant as an intermediary process can be recovered by the automated changes.
- Target file system: if the source and target file systems are similar (for example, going from IMAGE to an IMAGE emulator), then there would be less reason to use Axiant.
- Target platform: if you are migrating to UNIX, Cognos provides several UNIX command scripts to do the same actions as some of the Axiant migration tools.
- Target UI: if you are migrating to a Windows GUI (using Axiant), the Axiant migration tools have the added benefit of creating the Axiant objects for your application.
- Conditional compiles: If your applications require conditional compiles, using Axiant might complicate the migration. The conditional compiles would have to be removed before the migration and added back in after the migration was complete.

Information on planning a migration, both with and without the Axiant migration tools, can be found in "The Migration Plan" (p. 63).

Choosing the Migration Timing

When you plan any operation with multiple changes, the cleanest process is usually to make one change at a time. This allows focus and fast identification of any unexpected issues that arise. For example, if you have multiple PowerHouse applications that are independent, you should migrate them independently.

When you migrate a PowerHouse application, the three possible migration types are to another platform, to another file system, and to another UI. Thus the 'divide and conquer' approach would be to perform each migration separately.

You can migrate to the terminal UI on all platforms for all file systems. We recommend that you postpone migration from one UI to another until after the platform and file system migrations have been completed. This guide thus focuses on platform and file system migrations. For further information on choosing an end-user UI, see the Axiant and PowerHouse Web documentation sets.

Can you similarly isolate the platform and file system migrations? That depends on costs, time pressures, and whether the two migrations are tied to one another. Let's consider a few scenarios.

Scenario A: Keeping the Same File System Type

This is the situation where you migrate to the file system recommended in the 'Simplest Migration' column of the table on (p. 12). An example is migrating from IMAGE to an IMAGE emulator. You essentially only have one migration, the platform move.

Scenario B: Changing File System Types

Examples of this situation include IMAGE or KSAM to Relational, and IMAGE to Indexed.

The ideal process would be to isolate the two migrations. If your source file system is IMAGE and your target file system is relational, you can first migrate to the target platform and an IMAGE emulator, then migrate from the IMAGE emulator to your relational database. Or you can migrate to the target platform and an indexed file system, then migrate from indexed to relational.

If migrating from IMAGE to Indexed, you can first migrate from IMAGE to the target platform and an IMAGE emulator, then migrate from the IMAGE emulator to your indexed file system.

The most common reasons for not following this approach are license costs, time pressures, and company standards.

Scenario C: Merging File Systems

Every file system has its own issues to adapt to. If you have multiple file systems in your MPE/iX application, you may consider merging them all to one file system before migrating to the new platform.

For example, if your target file system is an IMAGE emulator, you may want to move your KSAM files to IMAGE while still on MPE/iX. This simplifies the combined platform-and-file-system migration. However it adds to the migration time and costs. For these reasons, this scenario is rare.

Moving Forward

"The Target Platform: UNIX" (p. 15) describes the changes you may have to make in your PowerHouse application when moving to UNIX.

"The Target Platform: Windows" (p. 27) describes the changes you may have to make in your PowerHouse application when moving to Windows.

"The File Systems" (p. 39) describes the changes you may have to make in your PowerHouse application when changing to the various file systems on UNIX or Windows.

"The Migration Plan" (p. 63) describes the 12-step plan to actually perform the migration. This identifies everything from identifying training needs to editing data definitions.

Chapter 2: The Target Platform: UNIX

Overview

This chapter identifies the differences between the MPE/iX and UNIX platforms that have to be addressed during the migration of a PowerHouse application. It also identifies the differences within PowerHouse between the two platforms.

For information on differences specifically due to a change in file systems, see "The File Systems" (p. 39).

Many of the differences between platforms require changes to file names as part of the migration. The Axiant migration tools can perform many of these changes automatically. For information on this and other aspects of actually moving your application from MPE/iX to UNIX, see "The Migration Plan" (p. 63).

The command interface on UNIX is known as a Shell. Two of the most popular shells are the Bourne Shell and the C Shell. PowerHouse works equally under either shell. The only difference is that you'll use different shell commands within your command scripts (which replace UDCs), depending on the shell you are running under.

Platform Differences

Groups and Accounts vs. Directory Paths

UNIX organizes its files in a hierarchy of directories (also known as folders). This is very similar to the Hierarchical File System (HFS) available in POSIX on MPE/iX. However, PowerHouse on MPE/iX does not support HFS. Traditional MPE/iX systems organize files in groups within accounts instead (essentially an HFS of only two levels).

In general, you create a top-level directory for each PowerHouse application, with subdirectories for different parts of your application. The simplest directory structure would be one that parallels your MPE/iX structure, with a subdirectory for each group.

All references to qualified file names (such as data files, command files, print files, or other types) within PowerHouse applications must be changed to match the directory structure. Also note that the UNIX format is "directory/subdirectory/file", which is the reverse order of the MPE/iX format, "file/group/account". For example,

```
COMMAND "QUICK AUTO=NITERUN.SALES.CORPORATE"
```

becomes

```
COMMAND "quick auto=/corporate/sales/niterun"
```

File Names

Case Sensitivity

Unlike MPE/iX, the UNIX operating system differentiates between lowercase and uppercase. Therefore any references to physical file names within your dictionary, source code, and command files must be consistent with the physical file name.

For example, the file names MYFILE, Myfile and myfile do not refer to the same physical file under UNIX.

A UNIX standard is for all names to be lowercase. We recommend that you eventually follow this practice, renaming your files to lowercase. However, you may wish to do this at some future time, after your migration is successful. This reduces the number of mismatches by accidentally referencing a file with the incorrect case. (

Tip: Since UNIX operating system commands are generally in lower case, most work on UNIX is done with the keyboard Caps Lock key turned off.

For information on how case sensitivity affects PowerHouse, see (p. 21).

Special Characters

Traditional MPE/iX file names consist of only letters and numbers, so there are no issues in porting these to UNIX. You may optionally add periods, underscores and hyphens within UNIX file names for clarity.

UNIX File Extensions

Traditional MPE/iX standards are to distinguish files of different types by either placing them in different groups (such as NITERUN.SOURCE vs. NITERUN.COMPILED) and/or giving the filenames different terminating characters (such as NITERUN vs. NITERUNC).

UNIX uses file extensions to identify file types (such as nightrun.qts vs. nightrun.qtc). These are generally three characters long but not necessarily so. There are some exceptions (such as scripts, the UNIX equivalent of UDCs, which usually have no extensions).

For more information, see "PowerHouse File Extensions" (p. 19).

Length

MPE/iX file names are restricted to eight characters in length. All flavors of UNIX allow more than that, usually 128 or 255, so you do not have to shorten any file names.

Similarly, MPE/iX pathnames (for example, `NIGHTRUN.SALES.CORPORATE`) are restricted to 26 characters in length (with an additional eight characters if you have a lockword), whereas UNIX pathnames allow up to 1023 characters.

File Equations

MPE/iX applications, including PowerHouse, make frequent use of file equations to point applications to files of different names or locations without editing the application. Environment variables are the equivalent on UNIX.

You need to define these environment variables to match your MPE/iX equivalents, usually defined within UDCs. For information on converting UDCs, see "Operating System Commands" (p. 16).

For information on the file equations for PowerHouse designated files (such as the default Powerhouse resource file), see "Designated Files and Environment Variables" (p. 20).

Operating System Commands

MPE/iX OS commands can be found in UDCs that you execute:

- outside PowerHouse
- from within QDESIGN COMMAND statements and RUN COMMAND verbs
- in QTP and QUIZ programs using SET JOB (prefixed with ":" or "!")

A great deal of processing power is contained in operating system command files. You must translate all UDCs you have created into UNIX scripts. Individual OS commands also have to be translated. (PowerHouse UDCs provided by Cognos already have the equivalent scripts provided on UNIX.)

If you have a large quantity of UDCs, an alternative is to purchase a third party MPE/iX emulator on UNIX.

Before translating UDCs, you should assess whether they need to come forward at all. A quick review of an application may turn up hundreds of such files. However, a detailed examination may find that most of them address MPE/iX requirements that are no longer applicable in a UNIX environment. Almost all applications can leave some UDCs behind.

UDCs that must come forward require that a few changes be made repeatedly to each file. This can best be handled through editor scripts (on either platform) or through UNIX utilities such as PERL. The key is to understand the changes and identify repetitive tasks. For more information, see "The Migration Plan" (p. 63).

Some Common Commands

Task	MPE/iX	UNIX
Shorten command names using an alias	command files or UDCs	`alias`
Change working area	`CHDIR`	`cd`
Copy files	`COPY, FCOPY, DSCOPY`	`cp`
Display all or part of text files	`PRINT`	`cat, more, tail`
Determine disk usage	`DISKUSE, REPORT`	`du`
Search for text patterns in a file	Search commands in editors	`grep, egrep, fgrep`
Terminate a process or job that is running	`ABORTJOB, ABORT`	`kill`
Rename files	`RENAME`	`mv`
Display current working directory	`SHOWVAR HPCWD`	`pwd`
Delete files	`PURGE`	`rm`
Delete directories	`PURGEDIR`	`rmdir`
Shorten file names using a file equation	`FILE`	`setenv`
Sort information	Sort/Merge utility	`sort`
Edit text or program files	`EDIT/3000`	`vi, ed, ex`
Display a list of who is logged on the system	`SHOWJOB, SHOWPROC`	`who`

An Example: File Equations

File equations are commonly used in MPE/iX PowerHouse applications to avoid hard-coding group and account name. This makes the application more portable.

The UNIX equivalent is an environment variable, defined with the `setenv` command. For example,

```
FILE PAY=PAY.ARCHIVE
```

would be replaced with

```
setenv PAY /archive/pay
```

An Example: A Batch Job

An MPE/iX UDC, QUIZRUN01	The UNIX Equivalent, quizrun01
`!JOB SOMEJOB,USER.GROUP`	
`:QUIZ`	`quiz <<eof`
`ACCESS EMPLOYEES`	`ACCESS EMPLOYEES`
`CHOOSE EMPLOYEE PARM`	`CHOOSE EMPLOYEE PARM`
`REPORT ALL`	`REPORT ALL`
`GO`	`GO`
`1`	`1`
`2`	`2`
`EXIT`	`EXIT`
`!EOJ`	`eof`
To execute:	To execute:
`STREAM QUIZRUN01`	`batch quizrun01`

In the UNIX script, `<<eof` redirects the command input to the rest of the script until encountering the eof string.

INFO=

On MPE/iX, parameters may be provided after the `INFO` Run command parameter. For example:

```
QTP INFO='AUTO=WEEKLYT NOLIST'
```

On UNIX, the "INFO=" string and quotes must be removed:

```
qtp AUTO=WEEKLYT NOLIST
```

SETCATALOG vs. Path

On MPE/iX, multiple UDCs are generally located in a single UDC file. For example, `:SETCATALOG MGRUDC` points to the MGRUDC file holding several manager UDCs.

On UNIX, scripts are located in separate files, allowing them to be grouped into separate directories. An environment variable named 'path' indicates which directories should be searched through when the user (or a program) invokes the name of a script.

Be sure to modify the path variable for your user accounts to point to the directories containing your scripts. For example, `set path = ($path /mgr_scripts/)` if you are using the UNIX Bourne shell, or `setenv PATH $PATH:/mgr_scripts/` if you are using the UNIX C shell.

System Management

The change in platforms also means changes in application management. In particular, review the methods and policies currently used for:

- application and data security
- backup, recovery, and data distribution
- software update distribution
- user interface
- end-user support
- capacity planning and performance management

File Access Control

The simplest level of application and data security is the UNIX file access control.

- Every user on a Unix system has a unique username, and is a member of at least one group.
- UNIX directories have access control properties that determine who can read, write, or execute the files within them. Each file similarly has access control properties determining read, write, and execute privileges.

- For both directories and files, you can control access at three levels: what your privileges as the file owner are, what other members of your group can do, and what others outside your group ('the world') can do.

You can likely map your MPE/iX file access control directly to the UNIX equivalent (for example, MPE/iX group access to UNIX group access). For information on how the different file access control maps to PowerHouse syntax, see (p. 21).

PowerHouse Differences

This discussion identifies differences between PowerHouse on MPE/iX and PowerHouse on UNIX, with the exception of differences relating to the file systems.

For information on differences due to a change in file systems, see "The File Systems" (p. 39).

Note: Many of the differences between platforms require changes to file names as part of the migration. The Axiant migration tools can perform many of these changes automatically. For information on this and other aspects of actually moving your application from MPE/iX to UNIX, see "The Migration Plan" (p. 63).

PowerHouse File Extensions

Traditional MPE/iX standards are to distinguish files of different types by either placing them in different groups (such as NITERUN.SOURCE vs. NITERUN.COMPILED) and/or giving the filenames different terminating characters (such as NITERUN vs. NITERUNC).

UNIX uses file extensions instead (such as nightrun.qts vs. nightrun.qtc). These are generally three characters long but not necessarily so. There are some exceptions (such as scripts, the UNIX equivalent of UDCs, which have no extensions as a general rule).

PowerHouse on UNIX uses the following naming conventions:

Component	Source	Compiled
Screens	name.qks	name.qkc
Reports	name.qzs	name.qzc
Runs	name.qts	name.qtc
PDL	name.pdl	name.pdc
Subfiles (data)	name.sf, name.ps	N/A
Subfiles (dictionary)	name.sfd, name.psd	N/A
QKGO	name.qkg	N/A
QSHOW	name.qss	N/A
QUTIL	name.qus	N/A

For a complete list of designated files and their names, see the *PowerHouse Rules* document.

For information on the timing and methods for renaming your files, see "The Migration Plan" (p. 63).

If you have renamed your files to use file extensions, you should also change all PowerHouse program calls to include the file extension. For example, the following command looks for NITERUN.qts first and then, if it cannot find it, NITERUN.qtc:

```
RUN COMMAND "qtp AUTO=NITERUN"
```

Running the source file can reduce performance, and it can cause errors if the source file does not include a `GO`. It may also display your source code (if you omit `NOLIST`). If your intent was to run the compiled program, change the call to:

```
RUN COMMAND "qtp AUTO=NITERUN.qtc"
```

Subfiles

On MPE/ix, subfile dictionary information is stored in the header area of each subfile file. On UNIX, however, the dictionary information is stored in a separate file. Thus each *name*.sf has a corresponding *name*.sfd. The same is true for portable subfiles, *name*.ps and *name*.psd. Portable subfiles have their data written in an ASCII format, which facilitates porting them between platforms.

When a program has `ACCESS *subfile`, you use the name of the dictionary file and you omit the extension. This usually doesn't matter if the data and dictionary portions have the same name, but they may be different.

On MPE/iX, the QTP code `SUBFILE PAYSF KEEP PORTABLE INCLUDE PAY` creates a subfile dictionary file named PAYSF and a subfile data file named PAYSFQ. On other platforms, including UNIX, the same code creates a subfile dictionary file named PAYSF.psd and a subfile data file named PAYSF.ps.

When moving portable subfiles from MPE/iX to UNIX, you must ensure that the tool transferring the files (such as FTP) does not insert a linefeed character at the end of each record. For more information, see "The Migration Plan" (p. 63).

Subfile Locations and File Domains

On MPE/iX, temporary file domains are used to hold temporary subfiles. UNIX does not have file domains, so starting a PowerHouse component session creates a temporary directory, PHnnnnn.TMP, to hold temporary subfiles and files. This directory is deleted when the PowerHouse component session ends. This directory is unique for each logon session, so two users running the same multi-pass report or multi-pass run do not read each other's temporary subfiles.

However, temporary subfiles on UNIX are deleted when you exit the component. On MPE/iX, they are only deleted when you sign off (when the session or job ends). So an application that writes a temporary subfile in QTP then exits QTP and tries to read that subfile in QUIZ will work on MPE/iX but not on UNIX.

A simple solution is to call the QUIZ program from within the QTP program:

```
ACCESS x
SUBFILE y INCLUDE ...
$QUIZ
  ACCESS *y LINK TO ...
  ...
```

The same solution using a compiled QUIZ report would be:

```
ACCESS x
SUBFILE y INCLUDE ...
$QUIZ AUTO=<path>\quizrun01.qzc
```

Designated Files and Environment Variables

Designated files are reserved for use as PowerHouse default files. Some of these names are reserved for use as file equations (MPE/iX) or environment variables (UNIX) by PowerHouse.

On UNIX, PowerHouse also locates designated files by searching for certain file extensions. For example, when you use the SAVE statement in QTP, PowerHouse creates a file containing the source code in the current directory and appends the file extension .qts to the filename. When PowerHouse searches for the file, it first searches for a filename that has the extension .qts appended.

For a list of PowerHouse designated files, their file extensions, and their environment variables, see the *PowerHouse Rules* document.

The Default Editor

The default editor used by a REVISE statement can be specified by the PHEDIT environment variable. This defaults to the vi editor on UNIX.

Case Sensitivity

As mentioned on (p. 15), the UNIX operating system differentiates between lowercase and uppercase, so any references to physical file names within your dictionary, source code, and command files must be consistent with the physical file name. In particular, note that PowerHouse executables and file extensions must be in lower case.

For example,

```
RUN COMMAND "QTP AUTO=NITERUN"
```

must become

```
RUN COMMAND "qtp auto=NITERUN.qtc"
```

or, more commonly,

```
RUN COMMAND "qtp auto=<path>/niterun.qtc"
```

The SYSTEM OPTIONS SHIFT as well as the SET <shift options> apply to relational column and tables but not physical file names. So the QDESIGN code, "SCREEN Test", results in a mixed-case file named Test.qkc on UNIX.

Interrupt Control

The user can interrupt prompting within PowerHouse components by pressing the interrupt keystroke in response to a prompt. On MPE/iX this is commonly Ctrl-Y; on UNIX this is commonly Ctrl-C.

PDL

Application Security Classes

The APPLICATION SECURITY CLASS and ASC statements on MPE/iX assign security with the LOGONID method. On UNIX, this must be changed to the UIC method:

```
ASC LOGONID user1
```

becomes

```
ASC UIC gid1,uid1
```

This also applies to the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement.

FILE

The following FILE statement options are MPE/iX-specific. They can be dropped without replacement, as they have no UNIX equivalents.

```
ASCII|BINARY, BLOCKING FACTOR n, CAPACITY n, REUSE
```

RECORD

The RECORD statement option `CAPACITY n` is MPE/iX-specific. It can be dropped without replacement, as it has no UNIX equivalent.

SYSTEM OPTIONS

The SYSTEM OPTIONS statement option `FLOAT IEEE|NONIEEE` is MPE/iX-specific. It can be dropped without replacement, as it has no UNIX equivalent.

Be aware that you may face a loss of data when migrating any floating point numbers from one platform to another. This is due to a loss of precision or significance; for example, 0.20 may be stored as 0.199999....

For more information on floating point data loss, see the *PowerHouse Rules* document.

QDESIGN/QUICK

DO EXTERNAL

Your application system may use 3GL programs (such as COBOL, Fortran, or C) either at the OS level or from within PowerHouse DO EXTERNAL calls. As with command files, the first thing to do is assess whether you really need to bring these forward. They may be vestiges of older systems, no longer needed.

Rewriting 3GL programs in PowerHouse code isn't recommended, especially if they perform complex mathematical calculations. Instead, find a compiler for the new UNIX environment. If you can't find a compiler, consider rewriting the programs in C or C^{++}.

You can remove the following options from your DO EXTERNAL calls on UNIX, because they are MPE/iX-specific:

- [CM|INTRINSIC|NM]
- [INPUT B|C|SAME]

To link to the external routine, QUICK creates a distinct process. A script named BuildExternal is included with PowerHouse on UNIX. BuildExternal is a C Shell (csh) script that takes routine names, object files, and an optional executable (driver) name and builds a program to communicate with QUICK in order to accomplish processing. For more information, see the *QDESIGN Reference* document.

QKGO

To start QKGO on UNIX, enter the `qkgo` command. On MPE/iX, this was `SETQKGO`.

If terminal (TIC) information is specified in QKGO, MPE/iX creates a context-binding file (for example, FILENAMEB) and a key sequence file (for example, FILENAMEK) and one or more terminal-group files (for example, FILENAMET). On UNIX, two files are created for all three of these cases: filenameb.dat and filenameb.idx, filenamek.dat and filenamek.idx, and filenamet.dat and filenamet.idx. PowerHouse must be licensed for C-ISAM data access.

Block Mode

Block mode is an MPE/iX feature, not supported on UNIX. The following statement options and procedural code must be removed:

```
SCREEN name BLOCKMODE
KEY x [NO]BLOCKTRANSFER
THREAD name INPUT B|C
RUN SCREEN|COMMAND|THREAD INPUT B|C|SAME
DO BLOB|EXTERNAL INPUT B|C|SAME
```

You may wish to consider Panel processing, although processing field by field is recommended. Panel processing is similar to Block mode processing, except it allows you to group fields into several panels rather than have the whole screen in one block. And since Panel processing is available on MPE/iX, you may wish to do this change before doing the platform migration. For more information on Panel processing, see the *QDESIGN Reference* document.

Function Keys

All platforms support multiple command processing and the KEY statement. So you can assign commands and command sequences to specific function keys on UNIX.

But only HP terminals (and HP terminal emulators) support function key labelling. So if your MPE/iX application defines a function key as a Shift key to map multiple levels of functions onto each physical key, your UNIX application will have no way to tell you what level you are on, and thus what function each key is mapped to.

The simplest solution is to use an HP terminal (or an HP terminal emulator) on UNIX. Otherwise, all function key usage depends on the terminal (or terminal emulator) being used.

Dynamic Screen Calling

With MPE/iX, you can back-reference file equations. This forces the file-equations to be re-evaluated every time they are referenced. For example, `SUBSCREEN *somescreen` references a previously existing file equation, `somescreen`.

With UNIX, this feature does not exist; symbolic links are not evaluated every time they are referenced. Instead, you can call a screen that has its name stored in a temporary item. For example, `SUBSCREEN ITEM somescreen` references a temporary item that has been set to a screen name.

An alternative is to use `SUBSCREEN $somescreen`, ensuring that a `RUN COMMAND` has previously done a `SETSYSTEMVAL("somescreen", "X")`.

Variables and Subprocesses

On MPE/iX, you can set a variable with a `RUN COMMAND "SETVAR ..."` and then reference it in a later `RUN COMMAND` subprocess. However on UNIX, a `RUN COMMAND "setenv ..."` only sets the environment variable for that one subprocess, so later `RUN COMMAND`s will not have that variable set.

The solution is to assign the environment variable with the `SETSYSTEMVAL` function and retrieve it with `GETSYSTEMVAL`. For information on these functions, see the *PowerHouse Rules* document.

QUICK Interactive Debugger

The QUICK Interactive Debugger generates debugging and listing files. On MPE/iX, these files are named *screen*D and *screen*L respectively. On UNIX, these files are named *screen*.qkd and *screen*.qkl. For more information, see the *QDESIGN Reference* document.

QTP

SET JOB

On MPE/iX, operating system commands prefixed with a colon (:) execute immediately. Operating system commands prefixed with an exclamation mark (!) are not executed immediately but are written to QTPSAVE.

On UNIX, operating system commands prefixed with a single exclamation mark (!) suspend the current process, and execute the Shell command. Operating system commands prefixed by two exclamation marks (!!) write the Shell command to the currently active save file with one exclamation mark. For examples, see the *QTP Reference* document.

Subfiles

The `SIZE n` option of the SUBFILE statement is not available on UNIX and must be removed.

Indexed subfiles require that PowerHouse be licensed for C-ISAM data access.

Intermediate Files

If a request on MPE/iX contains a sort phase, the designated file QTPSORT is used for an intermediate file. On UNIX, if a SORT is present, QTP creates two temporary work files, whose names incorporate the process ID. Note that since disk space is dynamic on UNIX, you do not need to pre-allocate disk space.

QUIZ

Subfiles

The "SIZE n" option of the SET SUBFILE statement is not available on UNIX and must be removed.

Indexed subfiles require that PowerHouse be licensed for C-ISAM data access.

Printer Options

The following statements are specifically for MPE/iX printers and should be removed for UNIX:

```
SET DEVICE CONTROL
SET REPORT NONSPOOLED [PRINTER]
SET REPORT FORMS
SET REPORT PRIORITY
```

Slave printing on MPE/iX may use HPSLAVE output blocks. On UNIX, ’`SET REPORT DEVICE PRINTER "slave.txt"`’ refers to a slave.txt script included with PowerHouse. For more information, refer to the *QUIZ Reference* document.

On MPE/iX, you often use file equations to control some printing characteristics, along with things like FORM. On UNIX, this can be simplified with SET REPORT DEVICE PRINTER <string>, where string is the specific device. For more information, see the *QUIZ Reference* document.

SET JOB

On MPE/iX, operating system commands prefixed with a colon (:) execute immediately. Operating system commands prefixed with an exclamation mark (!) are not executed immediately but are written to QUIZSAVE.

On UNIX, operating system commands prefixed with a single exclamation mark (!) suspend the current process, and execute the Shell command. Operating system commands prefixed by two exclamation marks (!!) write the Shell command to the currently active save file with one exclamation mark. For examples, see the *QUIZ Reference* document.

OMNIDEX

This statement is for OmniQuest, which is only available on MPE/iX. The same is true for the [NO]OMNIDEX options of the SET statement and the OMNIREUSE option of the GO statement.

After the Migration

After your application is up and running, you may want to take advantage of PowerHouse features on UNIX that were not available on MPE/iX. Some of these are identified below.

PDL

DATABASE, FILE

The "OPEN" filespec on UNIX may be an environment variable, which must be preceded by a dollar sign ($). This allows you to point to a different file name or location without changingyour PDL.This also applies to the PASSWORD and USERID options on the DATABASE statement.

Indexes

The INDEX and SEGMENT statements for C-ISAM files on UNIX allow a DESCENDING option. This indicates the order in which segments are stored in the index.

QDESIGN/QUICK

QDESIGN Locking

The QDESIGN SCREEN statement on MPE/iX supports LOCK RECORD FOR UPDATE. In addition, the same statement on UNIX supports LOCK RECORD FOR FIND.This can provide a more secure and efficient capability for applications that require a high degree of reliability. For more information, see the *QDESIGN Reference* document.

Rollback Clear

For relational applications, this QKGO setting specifies whether a rollback condition causes an immediate rollback or a ’rollback pending’ state. For more information, see the *PowerHouse and Relational Databases* document.

QTP

EXECUTE, REQUEST, SET

On UNIX, the INPUT LIMIT and PROCESS LIMIT options on the EXECUTE and REQUEST statements support a NOLIMIT value. So do the SET INPUT LIMIT and SET PROCESS LIMIT statements. Even when you use NOLIMIT, the maximum number of transactions processed is 2,147,483,647.

Locking

When indexed, sequential, and direct files are opened for exclusive access, no further locking strategy is required. As well, files opened for read-only access (READ, SHARE) are not locked on UNIX, whereas they are on MPE/iX. This may allow you to remove unnecessary code from your MPE/iX source files.

SUBFILE

The INDEX and SEGMENT options of the SUBFILE statement for indexed subfiles allow the DESCENDING option. This indicates the order in which segments are stored in the index.

QUIZ

SET SUBFILE

The INDEX and SEGMENT options of the SET SUBFILE statement for indexed subfiles allow the DESCENDING option. This indicates the order in which segments are stored in the index.

SET REPORT NOLIMIT

On UNIX, the SET REPORT LIMIT n statement has an alternative, SET REPORT NOLIMIT. Even when you use NOLIMIT, the maximum number of record complexes processed is 2,147,483,647.

Moving Forward

"The File Systems" (p. 39) describes the changes you may have to make in your PowerHouse application when changing to the various file systems on UNIX or Windows.

"The Migration Plan" (p. 63) describes the 12-step plan to actually perform the migration. This identifies everything from identifying training needs to editing data definitions.

Chapter 3: The Target Platform: Windows

Overview

This chapter identifies the differences between the MPE/iX and Windows platforms that have to be addressed during the migration of a PowerHouse application. It also identifies the differences within PowerHouse between the two platforms.

For information on differences specifically due to a change in file systems, see "The File Systems" (p. 39).

Many of the differences between platforms require changes to file names as part of the migration. The Axiant migration tools can perform many of these changes automatically. For information on this and other aspects of actually moving your application from MPE/iX to Windows, see "The Migration Plan" (p. 63).

Platform Differences

Groups and Accounts vs. Folders

Windows organizes its files in a hierarchy of folders within physical or logical disk drives. This is very similar to the Hierarchical File System (HFS) available in POSIX on MPE/iX. However PowerHouse on MPE/iX does not support HFS. Traditional MPE/iX systems organize files in groups within accounts instead (essentially an HFS of only two levels).

In general, you create a top-level folder for each PowerHouse application, with subfolders for different parts of your application. The simplest folder structure would be one that parallels your MPE/iX structure, with a subdirectory for each group.

All references to qualified file names (such as data files, scripts, print files, or other types) within PowerHouse applications have to be changed to match the folder structure. Also note that the Windows format is "drive:\folder\subfolder...\file", which is the reverse order of the MPE/iX format, "file/group/account". For example,

```
COMMAND "QUICK AUTO=NITERUN.SALES.CORPORATE"
```

would become

```
COMMAND "QUICK AUTO=C:\CORPORATE\SALES\NITERUN"
```

File Names

Case Sensitivity

MPE/iX is not case sensitive and uses upper case. The Windows operating system uses mixed case, however it is not case sensitive. In other words, TEST.qkc and test.qkc are considered the same name. For more information, see "PowerHouse File Extensions" (p. 31).

Special Characters

Traditional MPE/iX file names consist of only letters and numbers, so there are no issues in porting these to Windows. You may optionally add periods, underscores and hyphens within Windows file names for clarity.

Windows File Extensions

Traditional MPE/iX standards are to distinguish files of different types by either placing them in different groups (such as NITERUN.SOURCE vs. NITERUN.COMPILED) and/or giving the filenames different terminating characters (such as NITERUN vs. NITERUNC).

Windows uses file extensions to identify file types (such as NITERUN.QTS vs. NITERUN.QTC). These are generally three characters long but not necessarily so.

For more information, see "PowerHouse File Extensions" (p. 31).

Length

MPE/iX file names are restricted to eight characters in length. Windows allows 256 characters, so you do not have to shorten any file names.

Similarly, MPE/iX path names (for example, `NIGHTRUN.SALES.CORPORATE`) are restricted to 26 characters in length (with an additional eight characters if you have a lockword), whereas Windows path names allow up to 1023 characters.

File Equations

MPE/iX applications, including PowerHouse, make frequent use of file equations to point applications to files of different names or locations without editing the application. Environment variables are the equivalent on Windows.

You need to define these environment variables to match your MPE/iX equivalents, usually defined within UDCs. For information on converting UDCs, see "Operating System Commands" (p. 28).

For information on the file equations for PowerHouse designated files (such as the default Powerhouse resource file), see "Designated Files and Environment Variables" (p. 32).

Operating System Commands

MPE/iX OS commands can be found in UDCs that you execute:

- outside PowerHouse
- from within QDESIGN COMMAND statements and RUN COMMAND verbs
- in QTP and QUIZ programs using SET JOB (prefixed with ":" or "!")

A great deal of processing power is contained in operating system scripts. Any UDCs you have created have to be translated into Windows scripts (also known as batch files or batch programs). Windows scripts can be executed from within a Command Prompt window or from the Run action of the Start Menu. Individual OS commands also have to be translated. (You don't need to translate the PowerHouse UDCs provided by Cognos. These have been replaced on Windows by installed shortcuts.)

If you have a large quantity of UDCs, an alternative is to purchase a third party MPE/iX emulator on Windows.

Before translating UDCs, you should assess whether whether your UDCs need to come forward at all. A quick review of an application may turn up hundreds of such files. However, a detailed examination may find that most of them address MPE/iX requirements that are no longer applicable in a Windows environment. Almost all applications can leave some UDCs behind.

UDCs that must come forward require that a few changes be made repeatedly to each file. This can best be handled through editor or word processor scripts (on either platform). The key is to understand the changes and identify repetitive tasks. For more information, see "The Migration Plan" (p. 63).

Some Common Commands

Task	MPE/iX	Windows
Change working area	`CHDIR`	`CD`

Task	MPE/iX	Windows
Copy files	`COPY, FCOPY, DSCOPY`	`COPY`
Display all or part of text files	`PRINT`	`TYPE, MORE`
Terminate a process or job that is running	`ABORTJOB, ABORT`	`TASKKILL` (XP only)
Rename files	`RENAME`	`RENAME, REN`
Delete files	`PURGE`	`DELETE`
Delete folders	`PURGEDIR`	`RMDIR`
Shorten file names using a file equation	`FILE`	`SET`

An Example: File Equations

File equations are commonly used in MPE/iX PowerHouse applications to avoid hard-coding group and account name. This makes the application more portable.

The Windows equivalent is an environment variable. It can be defined locally with the `SET` command inside a script. For example,

```
FILE PAY=PAY.ARCHIVE
```

would be replaced with

```
SET PAY=C:\ARCHIVE\PAY
```

Alternatively, it can be defined at the system level by right-clicking My Computer, selecting Properties, selecting the Advanced tab, and selecting Environment Variables.

An Example: A Batch Job

An MPE/iX UDC, QUIZRUN01	The Windows Equivalent, QUIZRUN01.BAT
`!JOB SOMEJOB,USER.GROUP`	
`:QUIZ`	`QUIZ`
`ACCESS EMPLOYEES`	`ACCESS EMPLOYEES`
`CHOOSE EMPLOYEE PARM`	`CHOOSE EMPLOYEE PARM`
`REPORT ALL`	`REPORT ALL`
`GO`	`GO`
`1`	`1`
`2`	`2`
`EXIT`	`EXIT`
`!EOJ`	
To execute:	To execute:
`STREAM QUIZRUN01`	`QUIZRUN01.BAT` (or select Run from the Start Menu)

INFO=

On MPE/iX, parameters may be provided after the INFO Run command parameter. For example:

```
QTP INFO='AUTO=WEEKLYT NOLIST'
```

On Windows, the "INFO=" string and quotes must be removed:

```
QTP AUTO=WEEKLYT NOLIST
```

SETCATALOG vs. PATH

On MPE/iX, multiple UDCs are generally located in a single UDC file. For example, `:SETCATALOG MGRUDC` points to the MGRUDC file holding several manager UDCs.

On Windows, scripts are located in separate files, allowing them to be grouped into separate folders. A system-level environment variable named PATH indicates which folders should be searched through when the user (or a program) invokes the name of a script.

The PowerHouse install procedures place the PowerHouse install locations into the PATH variable. Hence you can call QUIZ, for example, by simply typing the command `QUIZ` from within any folder you may be working in within your Command Prompt window.

Be sure to modify the PATH variable for your user accounts to point to the folders containing your scripts.

System Management

The change in platforms also means changes in application management. In particular, review the methods and policies currently used for:

- application and data security
- backup, recovery, and data distribution
- software update distribution
- user interface
- end-user support
- capacity planning and performance management

File Access Control

The simplest level of application and data security is Windows file sharing.

- Every user on a Windows network has a unique username, and is a member of at least one group.
- You can grant or deny access to a folder or file by either Everyone (all network users) or specified groups.
- You can allow or deny either Read, Change, or Full Control access.

Windows also provides a Security tab on the Sharing and Security property sheet of every folder and file. This is meant for file access among users on the same machine, whereas File Sharing is for networked access.

PowerHouse Differences

This discussion identifies differences between PowerHouse on MPE/iX and PowerHouse on Windows, with the exception of differences relating to the file systems.

For information on differences due to a change in file systems, see "The File Systems" (p. 39).

Note: Many of the differences between platforms require changes to file names as part of the migration. The Axiant migration tools can perform many of these changes automatically. For information on this and other aspects of actually moving your application from MPE/iX to Windows, see "The Migration Plan" (p. 63).

The Command Prompt Window

Since Windows has a graphical user interface and PowerHouse uses a terminal interface, PowerHouse components run from within a Command Prompt window. (To start a Command Prompt session, select Run from the Start menu and enter CMD.EXE. Alternatively, click the Start menu and select All Programs, Accessories, and then Command Prompt.)

You can also start the PowerHouse components directly by clicking the Start menu and selecting All Programs, PowerHouse <version>, and then <component>. These shortcuts to the Start menu were defined as part of the PowerHouse install process.

Note that any temporary files or folders created in a Command Prompt window are not accessible by programs in a separate Command Prompt window, although permanent files are.

PowerHouse File Extensions

Traditional MPE/iX standards are to distinguish files of different types by either placing them in different groups (such as NITERUN.SOURCE vs. NITERUN.COMPILED) and/or giving the filenames different terminating characters (such as NITERUN vs. NITERUNC).

Windows uses file extensions instead (such as NITERUN.qts vs. NITERUN.qtc). These are generally three characters long but not necessarily so. PowerHouse saves its file extensions in lower case.

PowerHouse on Windows uses the following naming conventions:

Component	Source	Compiled
Screens	NAME.qks	NAME.qkc
Reports	NAME.qzs	NAME.qzc
Runs	NAME.qts	NAME.qtc
PDL	NAME.pdl	NAME.pdc
Subfiles (data)	NAME.sf, NAME.ps	N/A
Subfiles (dictionary)	NAME.sfd, NAME.psd	N/A
QKGO	NAME.qkg	N/A
QSHOW	NAME.qss	N/A
QUTIL	NAME.qus	N/A
Resource File	NAME.rc	N/A

For a complete list of designated files and their names, see the *PowerHouse Rules* document.

For information on the timing and methods for renaming your files, see "The Migration Plan" (p. 63).

If you have renamed your files to use file extensions, you should also change all PowerHouse program calls to include the file extension. For example, the following command looks for NITERUN.qts first and then, if it cannot find it, NITERUN.qtc:

```
RUN COMMAND "QTP AUTO=NITERUN"
```

Running the source file can reduce performance, and it can cause errors if the source file does not include a `GO`. It may also display your source code (if you omit `NOLIST`). If you intended to run the compiled program, change the call to:

```
RUN COMMAND "QTP AUTO=NITERUN.qtc"
```

Paths in File Names

On both MPE/iX and Windows, when you build a screen or run or report without specifying a path in the filespec, the compiled file (.q*c) is saved in the current location (where you run the program from).

On Windows, though, if you run a PowerHouse component from the default Start menu shortcuts, this location is the product install location, by default. You can change this location by modifying the "Start in" property of the component shortcut.

To avoid saving your files in the "Start in" location when you run a component from a shortcut, run the component from a Command Prompt window (in which you've moved to the desired location with a CD command).

Subfiles

On MPE/ix, subfile dictionary information is stored in the header area of each subfile file. On Windows, however, the dictionary information is stored in a separate file. Thus each name.sf has a corresponding name.sfd. The same is true for portable subfiles, name.ps and name.psd. Portable subfiles have their data written in an ASCII format, which facilitates porting them between platforms.

When a program has "`ACCESS *subfile`", you use the name of the dictionary file and you omit the extension. The file name of the data portion is stored within the dictionary file.

On MPE/iX, the QTP code "`SUBFILE PAYSF KEEP PORTABLE INCLUDE PAY`" creates a subfile dictionary file named PAYSF and a subfile data file named PAYSFQ. On Windows, the same code creates a subfile dictionary file named PAYSF.psd and a subfile data file named PAYSF.ps.

When moving portable subfiles from MPE/iX to Windows, you must ensure that the tool transferring the files (such as FTP) does not insert a line feed character at the end of each record. For more information, see "The Migration Plan" (p. 63).

Subfile Locations and File Domains

On MPE/iX, temporary file domains are used to hold temporary subfiles. Windows does not have file domains, so starting a PowerHouse component session creates a temporary folder, tempnnnnnn, to hold temporary subfiles and files. The temporary folder is created in the location specified by the PHTEMP environment variable. This folder is deleted when the PowerHouse component session ends. This folder is unique for each logon session, so two users running the same multi-pass report or multi-pass run do not read each other's temporary subfiles.

However, temporary subfiles on Windows are deleted when you exit the component. On MPE/iX, they are only deleted when you sign off (when the session or job ends). So an application that writes a temporary subfile in QTP then exits QTP and tries to read that subfile in QUIZ works on MPE/iX but not on Windows.

A simple solution is to call the QUIZ program from within the QTP program:

```
ACCESS x
SUBFILE y INCLUDE ...
$QUIZ
  ACCESS *y LINK TO ...
  ...
```

The same solution using a compiled QUIZ report would be:

```
ACCESS x
SUBFILE y INCLUDE ...
$QUIZ AUTO=<path>\QUIZRUN01.QZC
```

Designated Files and Environment Variables

Designated files are reserved for use as PowerHouse default files. Some of these names are reserved for use as file equations (MPE/iX) or environment variables (Windows) by PowerHouse.

On Windows, PowerHouse also locates designated files by searching for certain file extensions. For example, when you use the SAVE statement in QTP, PowerHouse creates a file containing the source code in the current folder and appends the file extension .qts to the filename. When PowerHouse searches for the file, it first searches for a filename that has the extension .qts appended.

For a list of PowerHouse designated files, their file extensions, and their environment variables, see the *PowerHouse Rules* document. For the default settings of these environment variables, see the *PowerHouse 4GL 8.4 New Features* document.

The Default Editor

The default editor used by a REVISE statement can be specified by the PHEDIT environment variable. This defaults to the Notepad editor on Windows.

Interrupt Control

The user can interrupt prompting within PowerHouse components by pressing the interrupt keystroke in response to a prompt. On MPE/iX this is commonly Ctrl-Y; on Windows this is commonly Ctrl-C.

PDL

Application Security Classes

The APPLICATION SECURITY CLASS and ASC statements on MPE/iX assign security with the LOGONID method. LOGONID also works on Windows, so no changes are necessary. This also applies to the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement.

FILE

The following FILE statement options are MPE/iX-specific. They can be dropped without replacement, as they have no Windows equivalents.

```
ASCII|BINARY, BLOCKING FACTOR n, CAPACITY n, REUSE
```

RECORD

The RECORD statement option `CAPACITY n` is MPE/iX-specific. It can be dropped without replacement, as it has no Windows equivalent.

SYSTEM OPTIONS

The SYSTEM OPTIONS statement option `FLOAT IEEE|NONIEEE` is MPE/iX-specific. It can be dropped without replacement, as it has no Windows equivalent.

Be aware that you may face a loss of data when migrating any floating point numbers from one platform to another. This is due to a loss of precision or significance; for example, 0.20 may be stored as 0.199999....

For more information on floating point data loss, see the *PowerHouse Rules* document.

QDESIGN/QUICK

The Command Prompt

Because QUICK on Windows runs within a Console window (the Command Prompt) rather than a terminal, the following areas of your application may be affected:

- handling of escape sequences
- use of fonts and line drawings
- adjusting the size of the QUICK window
- setting highlights and colors

These issues are discussed in the *PowerHouse 4GL 8.4 New Features* document.

DO EXTERNAL

Your application system may use 3GL programs (such as COBOL, Fortran, or C) either at the OS level or from within QUICK DO EXTERNAL calls. As with scripts, the first thing to do is assess whether you really need to bring these forward. They may be vestiges of older systems, no longer needed.

Rewriting 3GL programs in PowerHouse code isn't recommended, especially if they perform complex mathematical calculations. Instead, find a compiler for the new Windows environment. If you can't find a compiler, consider rewriting the programs in C or C^{++}.

The DO EXTERNAL call on Windows does not support the syntax "[CM|INTRINSIC|NM]". This is MPE/iX-specific. You can simply remove these options from your calls.

Similarly, the DO EXTERNAL call on Windows does not support the syntax "[INPUT B|C|SAME]". These options are for Block mode, which is MPE/iX-specific. You can simply remove these options from your calls.

Note: The DO EXTERNAL call syntax on Windows has two formats or calling conventions, named C and PASCAL. Check your 3GL provider to determine which of these two calling conventions to use in the DO EXTERNAL call. For more information, see the *PowerHouse 4GL 8.4 New Features* document.

QKGO

To start QKGO on Windows, click the Start Menu, PowerHouse <version>, and then QKGO. On MPE/iX, this was done with the SETQKGO command.

On MPE/iX, the QKGO Maintenance system can generate a Terminal Interface Configuration (TIC) file. A different TIC file is required for each terminal type because the key codes are different. On Windows, though, there is only one keyboard type for QUICK that corresponds to the PC keyboard. The QKGO system on Windows cannot generate TIC files. A default TIC file is provided in the <install folder>\qkgo\resource folder. This TIC file contains the key codes for all the keys and key combinations that can be mapped, as well as a set of default key mappings that correspond to the PCANSI terminal type as defined for PowerHouse on UNIX. It can be modified as required to satisfy application requirements. To access a TIC file, use either the PHTICRS or AXTICRS environment variable to point to the TIC file. It is recommended that you make a copy of the default TIC file before modifying it.For more information, see the *PowerHouse 4GL 8.4 New Features* document.

Block Mode

Block mode is an MPE/iX feature, not supported on Windows. The following statement options and procedural code must be removed:

```
SCREEN name BLOCKMODE
KEY x [NO]BLOCKTRANSFER
THREAD name INPUT B|C
RUN SCREEN|COMMAND|THREAD INPUT B|C|SAME
DO BLOB|EXTERNAL INPUT B|C|SAME
```

You may wish to consider Panel processing, although processing field by field is recommended. Panel processing is similar to Block mode processing, except it allows you to group fields into several panels rather than have the whole screen in one block. And since Panel processing is available on MPE/iX, you may wish to do this change before doing the platform migration. For more information on Panel processing, see the *QDESIGN Reference* document.

Function Keys

All platforms support multiple command processing and the KEY statement. So you can assign commands and command sequences to specific function keys on Windows.

But only HP terminals (and HP terminal emulators) support function key labelling. So if your MPE/iX application defines a function key as a Shift key to map multiple levels of functions onto each physical key, your Windows application has no way to tell you what level you are on, and thus what function each key is mapped to.

A solution is to rewrite your application to use the Shift, Control, and Alt keys to simulate multiple levels.

For example, suppose your MPE/iX application mapped F1 as the level-shifting key and mapped F2 as 'U' at level 1 or as 'UR' at level 2. You could change the mapping so that F2 is always mapped as 'U' but CTRL+F2 is mapped as 'UR'.

For information on keyboard mapping, see "Modifying TIC Files" in the *PowerHouse 4GL 8.4 New Features* document.

Dynamic Screen Calling

With MPE/iX, you can back-reference file equations. This forces the file-equations to be re-evaluated every time they are referenced. For example, "`SUBSCREEN *somescreen`" references a previously existing file equation, `somescreen`.

With Windows, this feature does not exist; symbolic links are not evaluated every time they are referenced. Instead, you can call a screen that has its name stored in a temporary item. For example, `SUBSCREEN ITEM somescreen` references a temporary item that has been set to a screen name.

An alternative is to use `"SUBSCREEN $somescreen"`, ensuring that a `RUN COMMAND` has previously done a `SETSYSTEMVAL("somescreen", "X")`.

Variables and Subprocesses

On MPE/iX, you can set a variable with a `RUN COMMAND "SETVAR ..."` and then reference it in a later `RUN COMMAND` subprocess. However on Windows, a `RUN COMMAND "SET ..."` only sets the environment variable for that one subprocess, so later `RUN COMMAND`s will not have that variable set.

The solution is to assign the environment variable with the `SETSYSTEMVAL` function and retrieve it with `GETSYSTEMVAL`. For information on these functions, see the *PowerHouse Rules* document.

QUICK Interactive Debugger

The QUICK Interactive Debugger generates debugging and listing files. On MPE/iX, these files are named *screen*D and *screen*L respectively. On Windows, these files are named *screen*.qkd and *screen*.qkl. For more information, see the *QDESIGN Reference* document.

QTP

SET JOB

On MPE/iX, operating system commands prefixed with a colon (:) execute immediately. Operating system commands prefixed with an exclamation mark (!) are not executed immediately but are written to QTPSAVE.

On Windows, operating system commands prefixed with a single exclamation mark (!) suspend the current process, and execute the command. Operating system commands prefixed by two exclamation marks (!!) write the command to the currently active save file with one exclamation mark. For examples, see the *QTP Reference* document.

Subfiles

The `SIZE n` option of the SUBFILE statement is not available on Windows and must be removed.

Indexed subfiles require that PowerHouse be licensed for DISAM data access.

Intermediate Files

If a request on MPE/iX contains a sort phase, the designated file QTPSORT is used for an intermediate file. On Windows, if a SORT is present, QTP creates two temporary work files, whose names incorporate the process ID. Since disk space is dynamic on Windows, you do not need to pre-allocate disk space.

QUIZ

Subfiles

The "SIZE n" option of the SET SUBFILE statement is not available on Windows and must be removed.

Indexed subfiles require that PowerHouse be licensed for DISAM data access.

Printer Options

The printer configuration for the current Windows session is used as the destination printer. To direct the Report to another printer, change your printer configuration using the Control Panel. See your Microsoft Windows documentation for information on printer configuration in Control Panel. Or if you are running QUIZ from a Command Prompt window, you can specify the printer through options on the Print command.

The following statements are specifically for MPE/iX printers and should be removed for Windows:

```
SET DEVICE CONTROL
SET REPORT NONSPOOLED [PRINTER]
SET REPORT FORMS
SET REPORT PRIORITY
```

Slave printing on MPE/iX may use HPSLAVE output blocks. On Windows, '`SET REPORT DEVICE PRINTER "slave.txt"`' refers to a slave.txt script included with PowerHouse. For more information, refer to the *QUIZ Reference* document.

On MPE/iX, you often use file equations to control some printing characteristics, along with things like FORM. On Windows, this can be simplified with SET REPORT DEVICE PRINTER <string>, where string is the specific device. For more information, see the *QUIZ Reference* document.

SET JOB

On MPE/iX, operating system commands prefixed with a colon (:) execute immediately. Operating system commands prefixed with an exclamation mark (!) are not executed immediately but are written to QUIZSAVE.

On Windows, operating system commands prefixed by two exclamation marks (!!) write the Shell command to the currently active save file with one exclamation mark. Operating system commands prefixed with a single exclamation mark (!) suspend the current process, and execute the Shell command. For examples, see the *QUIZ Reference* document.

OMNIDEX

This statement is for OmniQuest, which is only available on MPE/iX. The same is true for the [NO]OMNIDEX options of the SET statement and the OMNIREUSE option of the GO statement.

After the Migration

After your application is up and running, you may want to take advantage of PowerHouse features on Windows that were not available on MPE/iX. Some of these are identified below.

PDL

DATABASE, FILE

The "OPEN" filespec on Windows may be an environment variable, which must be preceded by a dollar sign ($). This allows you to point to a different file name or location without changingyour PDL.This also applies to the PASSWORD and USERID options on the DATABASE statement.

Indexes

The INDEX and SEGMENT statements for DISAM files on Windows allow the DESCENDING option. This indicates the order in which segments are stored in the index.

QDESIGN/QUICK

QDESIGN Locking

The QDESIGN SCREEN statement on MPE/iX supports LOCK RECORD FOR UPDATE. In addition, the same statement on Windows supports LOCK RECORD FOR FIND.This can provide a more secure and efficient capability for applications that require a high degree of reliability. For more information, see the *QDESIGN Reference* document.

Rollback Clear

For relational applications, this QKGO setting specifies whether a rollback condition causes an immediate rollback or a 'rollback pending' state. For more information, see the *PowerHouse and Relational Databases* document.

QTP

EXECUTE, REQUEST, SET

On Windows, the INPUT LIMIT and PROCESS LIMIT options on the EXECUTE and REQUEST statements support a NOLIMIT value. So do the SET INPUT LIMIT and SET PROCESS LIMIT statements. (Even when you use NOLIMIT, the maximum number of transactions processed is 2,147,483,647.)

Locking

When indexed, sequential, and direct files are opened for exclusive access, no further locking strategy is required. As well, files opened for read-only access (READ, SHARE) are not locked on Windows, whereas they are on MPE/iX. This may allow you to remove unnecessary code from your MPE/iX source files.

SUBFILE

The INDEX and SEGMENT options of the SUBFILE statement for indexed subfiles allow the DESCENDING option. This indicates the order in which segments are stored in the index.

QUIZ

SET SUBFILE

The INDEX and SEGMENT options of the SET SUBFILE statement for indexed subfiles allow the DESCENDING option. This indicates the order in which segments are stored in the index.

SET REPORT NOLIMIT

On Windows, the SET REPORT LIMIT n statement has an alternative, SET REPORT NOLIMIT. (Even when you use NOLIMIT, the maximum number of record complexes processed is 2,147,483,647.)

Moving Forward

"The File Systems" (p. 39) describes the changes you may have to make in your PowerHouse application when changing to the various file systems on Windows or Windows.

"The Migration Plan" (p. 63) describes the 12-step plan to actually perform the migration. This identifies everything from identifying training needs to editing data definitions.

Chapter 4: The File Systems

Overview

This chapter contains a separate section for each of the MPE/iX file systems: IMAGE, KSAM, MPE, and relational. In each section, we identify the target file systems available for the given source file system, and then discuss the work involved in migrating between the two file systems.

We discuss what must change during a migration in both the dictionary and the programs. Where differences are major, we discuss details of specific target file systems.

This chapter discusses the PowerHouse differences between file systems. For a discussion of the relative advantages of each file system, see "Migration Choices" (p. 11). For information on moving the application files between platforms and other aspects of the overall migration process, see "The Migration Plan" (p. 63). For more detailed information on how PowerHouse works with relational databases, see the *PowerHouse and Relational Databases* document or consider attending the *PowerHouse Relational Interface* training (Parts I and II).

Migrating from IMAGE

The most likely types of file systems you will migrate your IMAGE databases to are:

- IMAGE emulators
- relational databases
- indexed file systems

IMAGE to An IMAGE Emulator

Eloquence is a database system from Marxmeier that includes an IMAGE emulation layer. This layer allows applications to continue to use the IMAGE intrinsic calls as if they were interfacing with IMAGE. PowerHouse currently supports the Eloquence emulator on HP-UX and Windows. You can thus migrate to another platform with minimal changes to your applications.

Other IMAGE emulators exist that use a similar approach except that the underlying file system is a relational database. IMAGE emulators exist on all of the UNIX platforms that PowerHouse supports as well as Windows.

To convert your PowerHouse PDL to handle an Eloquence database in place of the original IMAGE database, simply change the FILE statement from TYPE IMAGE to TYPE ELOQUENCE, with an optional password change. (Or remove the TYPE clause, which is also optional.)

Sample Source PDL (IMAGE)	Target PDL (Eloquence)
Create Dictionary EMPDICT	Create Dictionary EMPDICT
Element ADDRESSCHARACTER &	Element ADDRESSCHARACTER &
; No changes are needed for	; No changes are needed for
; the Element definitions	; the Element definitions
...	...
FILE EMPDB &	FILE EMPDB &
Organization Database &	Organization Database &
Type IMAGE &	**Type Eloquence** &
Open EMPDB &	Open EMPDB &
Password 'WRITER'	**Password "password/dba"** &
Critical Item Update	Critical Item Update
Record CARDS &	Record CARDS &
Capacity 17 &	Capacity 17 &
Organization MASTER &	Organization MASTER &
; No changes are needed for	; No changes are needed for
; the Record or Item	; the Record or Item
; definitions	; definitions
...	...
LOAD	LOAD

The CAPACITY option on the FILE statement is not supported by Eloquence, but it can be left in the code and will only produce a compile warning.

The QUTIL code to create the new objects in Eloquence is simply:

```
qutil dict=EMPDICT
> create base EMPDB
> exit
```

Because no changes need to be made to your data structures, the changes to your PowerHouse application code are also minimal. The only changes necessary are those that reference external objects (such as OS commands and DO EXTERNALs). As this is platform dependent, these issues are discussed in "The Target Platform: UNIX" (p. 15) and "The Target Platform: Windows" (p. 27).

As an alternative to a full IMAGE emulator, tools are available which convert IMAGE schemas and data to a relational database environment. Some also offer an emulator of the IMAGE application interface. These could provide the opportunity to stay with an IMAGE look-alike while also using the company standard database (such as Oracle).

IMAGE to Relational

PowerHouse currently supports the following relational database management systems (RDBMSes).

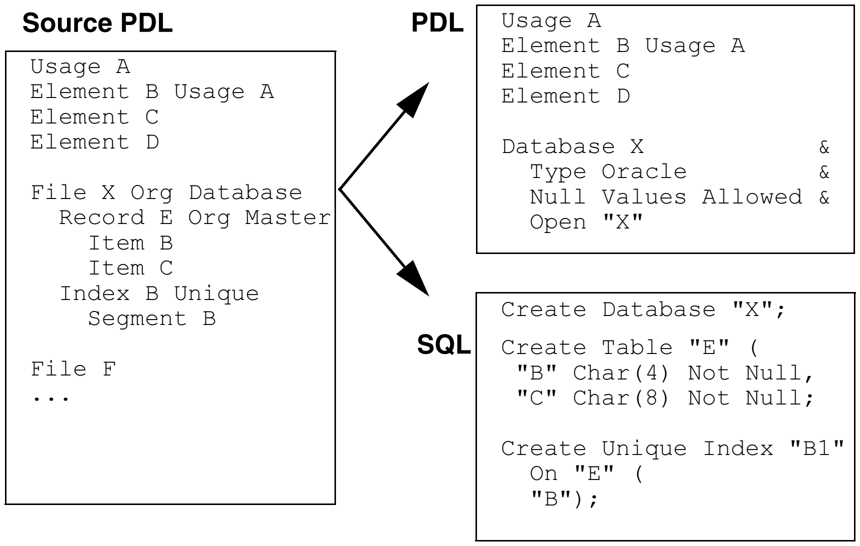
UNIX	Windows
Oracle	Oracle
Sybase	Sybase
IBM DB2 UDB	IBM DB2 UDB
DataDirect ODBC to SQL Server	MS SQL Server 7 and 2000 (via ODBC)
ALLBASE/SQL (HP-UX only)	

Relational databases have many automated features, such as query optimization and database integrity processing. They also offer many administration features to simplify maintenance. They can handle very large amounts of data. And they can improve performance, if you modify your application to use cursors (which process large sets of data at a time).

You need to make several types of changes to your application source files, both data definitions and programs. This can be done manually, in an editor, or it can be done with automated tools. For information on these tools, see "The Migration Plan" (p. 63).

Data Definitions

Your PDL file for IMAGE defined both logical objects (such as Elements) and physical storage objects (such as Files, Records, Items and Indexes). The PDL for a relational application contains only the logical objects and a *pointer* to the database. You also create a database SQL source file containing the SQL equivalent of your File, Record, Item and Index definitions.



```
Usage A
Element B Usage A
Element C
Element D

File X Org Database
  Record E Org Master
    Item B
    Item C
  Index B Unique
    Segment B

File F
...
```

```
Usage A
Element B Usage A
Element C
Element D

Database X             &
  Type Oracle          &
  Null Values Allowed  &
  Open "X"
```

```
Create Database "X";
Create Table "E" (
 "B" Char(4) Not Null,
 "C" Char(8) Not Null;

Create Unique Index "B1"
  On "E" (
  "B");
```

The PDL Database statement often includes database OWNER, USERID and PASSWORD options.

In addition to Usage, Element, and Database statements, the PDL may also contain Application Security Class statements. PERMIT statements cannot be applied to relational data, but ASCs may still be used in the USERS INCLUDE options of screens, runs, and reports.

Use QSHOW to generate the initial SQL from your dictionary, where `<file>` is any dictionary file, including an IMAGE database like X:

```
QSHOW
> SET LANGUAGE SQL
> GENERATE DATABASE <file>
```

This creates a text file, QSHOSQL, which you then modify to include syntax specific to your chosen RDBMS. Alternatively, Axiant can convert your entire dictionary, as described in "The Migration Plan" (p. 63).

Identifying the Database

When accessing a relational table within a PowerHouse component, use the IN clause to identify the database as defined within your PDL. For example:

QDESIGN: `FILE ORDERS` **`IN X`**

QUIZ/QTP: `ACCESS ORDERS` **`IN X`**

Alternatives are the SET DATABASE statement (to provide the default database) and the program parameter SUBDICT=SEARCH. If all table names are unique, then using SUBDICT means not having to change source code.

Naming Conventions: Hyphens to Underscores

Unlike IMAGE, relational databases do not permit hyphens ("-") within object names. This standard is to avoid confusion with the subtraction or minus character in operations.

The most common replacement character is an underscore ("_"). For example, "MONTH-NAME" becomes "MONTH_NAME". The change is a simple editor find-and-replace in your source files. However, do these one by one rather than with a wildcard, to avoid making the change in code performing a subtraction operation. Alternatively, the Axiant migration tools do this automatically.

Naming Conventions: Sybase Case Sensitivity

Sybase is case sensitive. However, PowerHouse upshifts object names by default. Hence, a Sybase table or column defined in mixed or lower case (for example, "Billings") would not be recognized by PowerHouse (where `ACCESS Billings` would become `ACCESS BILLINGS`).

There are several ways to avoid this issue. The simplest method is to disable the automatic upshifting and use the correct case in your PowerHouse programs. To disable the upshifting, add the following to your PDL:

```
SYSTEM OPTIONS SHIFT NOSHIFT
```

Normalization

RDBMSes achieve their benefits by working on the basis of normalized data. Normalization is a design process removing redundancy in your data definitions. Thus a migration to relational requires data structure changes while moving from PDL to SQL.

Ideally, you would make a full review of your record structures and implement a fully normalized data model. At a minimum, it is necessary to remove all non-relational features from your data definitions. This may result in additional data objects, but the resulting application programs will be easier to understand and maintain.

The following designs are not normalized:

- item substructures
- arrays (occurring or repeating items)
- item redefinitions
- multiple record layouts (used for coded records with Select values)

Item Substructures

Two ways to normalize substructures are:

- remove the highest level item and use lower levels only

Unnormalized	Normalized

`Item MONTH-KEY`
`Begin Structure`
`  Item INITIALS`
`  Item MONTH`
`End Structure`
	`Item INITIALS`
	`Item MONTH`

- remove lower levels and use the highest level only

Unnormalized	Normalized

`Item TRANS-DATE`
`Begin Structure`
`  Item YEAR`
`  Item MONTH`
`  Item DAY`
`End Structure`
	`Item TRANS_DATE`

The first method is useful when the end user tends to enter the subitems individually; the second approach is useful when you commonly work with the whole structure at a time (for example, a date field).

If you choose the first method yet you need to access the higher-level item in your applications, you can create a concatenated item in either the application (for example, a Define) or the database (for example, a calculated column).

Similarly, if you choose the second method yet you need to access the lower-level item in your applications, you can extract it in either the application (for example, a Define) or the database (for example, a calculated column).

Arrays

Some RDBMSes support arrays, but they must be replaced with normalized structures for PowerHouse applications. Three ways to normalize arrays are:

- define individual fields for each occurrence

Unnormalized Sales File	Normalized

`Item PRODUCT-NO`
`ITEM MONTHLY-TOTALS &`
`  Occurs 12`
`...`
	`Item PRODUCT_NO`
	`Item JANUARY_TOTAL`
	`Item FEBRUARY_TOTAL`
	`...`

- create a detail table

Unnormalized Sales File	Normalized Tables

`Item PRODUCT-NO`
`ITEM MONTHLY-TOTALS &`
`  Occurs 12`
`...`
	Sales table:
	`Item PRODUCT_NO`
	`...`
	Monthly_Sales table:
	`Item PRODUCT_NO`
	`Item MONTH`
	`Item MONTHLY_TOTAL`

- define a single item as long as all the occurrences

  (This method is only available if the items are character or zoned unsigned.) For example, `Item DAY-CODE Char Size 2 Occurs 7` could become `Item DAY_CODE Char Size 14`. You would then use the [n:m] function in your applications to extract the desired day.

The first method is the simplest metadata change, but it is less practical if there are a large number of occurrences, and it is not true normalization. It can result in many lines of application code. For example, a screen cluster around one Field statement would be replaced by multiple Field statements; a procedural For loop would be replaced similarly. This method might be appropriate for MONTH (12 values) but not for DAY_OF_MONTH (31 values).

The second method may be less efficient when reading, as the detail table holds one record for each occurrence. However it can be more efficient in program processing logic, and require simpler program changes. The I/O loss is often recovered by both the CPU savings and the easier maintenance effort. As well, aggregate processing can be simpler and faster if you change your logic to use SQL.

Screen Example, Unnormalized Sales File	Screen Example, Normalized Tables

`Screen X`
`File SALES`
 	
`Field PRODUCT_NO`
`Cluster Occurs with &`
`  MONTHLY_TOTAL`
`Field MONTHLY_TOTAL`
`End Cluster`
`...`
	`Screen X`
	`File SALES` **`In DB`**
	`File MONTHLY_SALES In DB &`
	`  Detail Occurs 12`
	`Field PRODUCT_NO`
	`Cluster Occurs with &`
	`  MONTHLY_SALES`
	`Field MONTHLY_TOTAL`
	`End Cluster`
	`...`

Item Redefinitions

Two ways to normalize Redefines are:

- to remove one item and change program logic to only use that one

Unnormalized	Normalized

`Item USERID`
`Item USERIDCHAR &`
`  Redefines USERID`
	`Item USERID`

- to keep separate items and maintain them in your program logic

Unnormalized	Normalized

`Item USERID`
`Item USERIDCHAR &`
`  Redefines USERID`
	`Item USERID`
	`Item USERIDCHAR`

Which method you use will depend on the reason for the original Redefine. The second method need not require changes to your application code if you use RDBMS triggers or computed fields to automatically populate one item with the value of the other item.

Multiple Record Layouts

Multiple record layouts for a single file are usually (but not necessarily) used when creating coded records with Select clauses. For example:

```
File CONTACTS
  Record CONTACTS
    Item RECTYPE
    ...
  Record SUPPLIERS
    Item RECTYPE SELECT "S"
    ...
```

Two ways to normalize multiple record layouts are:

- to use one table and add views to separate the data
- to create separate tables and merge them with a view

If you use a single large table, you can create views that report from specific subsets of that data. Depending on the RDBMS and the Select statement within the view, you may also be able to write to the table through the view.

If you choose to use separate tables, you may be able to report from all tables at once by creating a database view that includes the data from all the tables.

Program changes will likely be necessary after removing coded records. This includes changing or removing Select options and "Set File Open n" statements.

Primary Keys and Indexes

A primary key is a unique identifier of each record in the table. Some RDBMSes, such as Sybase, require a primary key in every table definition. Regardless of the RDBMS, we recommend you add a primary key and a unique index to every table if they do not already have one. This maintains data integrity. Equally important, it allows table records to be updated or deleted within Cursors. Cursors are a relational feature which can greatly improve performance.

Primary keys and indexes should be modified if they contain an item which was a substructure and you have replaced the substructure with the lower-level items (p. 42). The key or index must have its segments defined with the same items.

Some RDBMSes, such as Oracle and DB2, automatically generate a unique index based on the primary key for every table. You may want to disable this database feature if you have your own unique indexes already declared. (An alternative is to remove your own index definitions, but you must also change any programs that refer to those indexes through syntax like VIAINDEX.)

Index names must be unique within a relational database, so you may have to rename some indexes which are currently the same within different record structures. Don't forget to change any programs that reference the old index names.

Automatic Masters

An IMAGE automatic master is a dataset containing a key only and no other items. IMAGE maintains the automatic master records, deleting and adding as needed. Automatic masters are typically used for two purposes. First, they can provide an easy way to ensure unique values by providing a lookup key. Second, they can provide a convenient retrieval path.

Since relational databases can provide the same functionality through unique indexes, you no longer need the record structures defined as IMAGE automatic masters. Instead, add a unique index to the new detail table. If you remove an automatic master, also replace any code that references the automatic master directly with code that references the unique index in the detail table.

An automatic master provides the key corresponding to the detail dataset search item. While IMAGE does not allow you to declare detail search items as unique indexes, PowerHouse does allow this, and automatically generates a LOOKUP NOTON. In cases such as these, no code changes are required.

On the other hand, if a FIELD statement for a column of the detail table does a LOOKUP NOTON the automatic master to ensure uniqueness, then the LOOKUP NOTON must be done against the detail table itself using the unique index.

Regarding search retrieval paths, in most cases the retrieval is done against the detail dataset in PowerHouse code, so removing the automatic master and replacing it with an index doesn't usually require any code changes.

Manual Masters and Details

Relational databases make no distinction between tables used for master vs. detail data. When converting your PDL Record statements into SQL Table statements, you just drop the Master or Detail option.

Similarly, the `LINKS TO record [SORT ON item]` option of the Index statement has no equivalent, so it can be dropped too. Just be sure to keep (or add) an index for each table.

To maintain the data integrity offered by IMAGE masters and details, you can define foreign keys within the database. These are keys which link a master table to a detail table, preventing (a) insertion of a detail record with no corresponding master, and (b) deletion of a master record that still has details.

Note that foreign key constraint violations are identified by the database at update time. To avoid these violations in the first place, your QUICK and QTP programs should be reviewed to ensure that they (a) add master records before adding details, and (b) delete detail records before deleting the master. This can be explicitly controlled by the order of your QUICK Put verbs or QTP Output statements.

With QUICK, PUT statement order can be implicitly generated by a program parameter, "`update=fkc_put_order`" (which we recommend that you use if you haven't coded your own Update procedures). For more information, see the *PowerHouse 4GL 8.4 New Features* document.

With QTP, situations requiring that you add a master record before adding details can be resolved through an "`OUTPUT MASTER ADD AT START OF control-break`", followed by an ITEM statement to set the index value.

Ordered Retrieval

Records in a relational table are not stored in a specific order (for example, chronological or by unique index). Therefore any programs that expect the data in a specific order but which have no Sort statement will need to be changed.

In the following examples, the new code is shown in bold:

QUICK

```
FILE EMPLOYEES IN TRACKING
ACCESS ORDERED &
  VIA LAST_NAME USING LAST_NAME REQUEST LAST_NAME
```

QUIZ/QTP

```
ACCESS EMPLOYEES IN TRACKING
CHOOSE VIAINDEX LAST_NAME
```

For more information on ordering, see the Ordered, Orderby and Viaindex options in the *QDESIGN, QUIZ,* and *QTP Reference* documents.

Backwards Reads

IMAGE allows you to do a GET <file> BACKWARDS, in order to read the file in descending order without defining a separate index. The BACKWARDS option is not supported for relational databases, so you should modify your programs to read the table using an ORDERBY <column> DESCENDING.

Using the Backwards QUICK command (the backslash) is not supported for relational databases.

Null Values

Relational databases support the concept of null values by default. As this affects programming logic and thus requires evaluation of all your program code, you may wish to disable null support initially and investigate its benefits after your system has been fully migrated.

A null value is an unknown value. It indicates nothing, not zero or blank. Nulls model the real world better than binary processing (True or False) does, because the real world often has missing information.

For example, calculations containing null values evaluate to null. So Total = Supply + Labour evaluates to null if either Supply or Labour is null. If you don't know what the labour costs are, you don't know what the total cost is. (Compare this to binary logic, where Total = Supply if Labour has nothing in it, because 'nothing' would be treated as a zero.)

Null support can cause conversion issues when loading your MPE/iX data into a relational database. It can also cause logic errors in existing program code that isn't expecting null processing rules. For more information on null values in a PowerHouse context, see the *PowerHouse Rules* document as well as the training courses, *The PowerHouse Relational Interface (Parts I and II).*

Null value processing is disabled within PowerHouse by default: the PDL Database statement has the default option of 'Null Values Not Allowed'. This can be changed once the migration is complete and you are ready to upgrade your programs to support nulls.

Miscellaneous Syntax

The Lock and Unlock verbs, as well as the Lock option are generally not supported for relational tables. Allbase and ORACLE have limited support for them, and they are ignored for other database systems. In any case, for all RDBMSes, we recommend they be replaced with transaction control statements. For more information, see "The Next Step: Relational Features" (p. 47).

The creation and maintenance of your relational database is handled by database tools rather than PowerHouse. Thus, in QUTIL, the following syntax does not apply to relational tables, and must be removed: `CREATE|DELETE BASE x`

In QDESIGN, the following syntax does not apply to relational tables, and must be removed: `STARTLOG, STOPLOG, MEMOLOG`

Two PowerHouse options which behave differently in a relational application are Open numbers (for example, "FILE x Open 1") and the Close verb or option.

- Open numbers cause separate transactions to be created. We recommend they be replaced with transaction control statements.
- The Close verb or option commits a transaction and then detaches from the database. We recommend they be replaced with a Commit verb.
- For more information, see "The Next Step: Relational Features" (p. 47).

DBMODE syntax does not apply to relational tables, and must be removed:

- QUICK: `FILE x` **`OPEN DBMODE n`**
- QTP: `SET FILE X` **`DBMODE n [DEFERRED]`**
- QUIZ: `SET` **`DBMODE n`**

The Next Step: Relational Features

Once you have successfully migrated your application to a relational environment, your next step is to take advantage of features that were not available with IMAGE.

Some of these features can simplify your programs, some can extend your program functionality, and several of them can improve performance. Some of these features require changes to the database, while others require changes to your program code.

Some features to consider adding to the database are:

- triggers (SQL code inserted in the database to take place whenever any application inserts, modifies, or deletes data in a specified table);
- views (virtual tables, constructed in the database to merge or separate data from one or more tables)
- computed fields (columns whose definition includes a calculation, performed at write time once, which improves read time);
- varchars (variable character datatypes, which can be more efficient than fixed-length character items).

Some features to consider adding to your programs are:

- SQL cursors, essentially temporary views defined for the duration of that program run;
- SQL commands (Insert, Update, and Delete) that you can include within QUICK procedures and QTP calls;
- transaction control, the relational method of controlling data access;
- null value support, allowing your program logic to handle real-world situations of missing data;
- stored procedures, functions defined within the database which can be explicitly called by QUICK or QTP.

For more information, see the *PowerHouse and Relational Databases Reference* document and the training courses, *The PowerHouse Relational Interface (Parts I and II)*.

IMAGE to Indexed

Migrating from IMAGE to C-ISAM (UNIX) or DISAM (Windows) is essentially like migrating from IMAGE to KSAM but on a different platform. It is not as change-free as moving to an IMAGE emulator, but it is less involved than a relational migration.

Of course, the resulting application won't have all the system and performance benefits of an RDBMS (once the effort is taken to take advantage of relational features). However it can offer similar performance to that of an IMAGE emulator at a lower cost than either of the alternatives.

Data Definitions

Edit your PDL to move the Record, Item, Index and Segment statements from the IMAGE File statement(s) into separate File statements, and change the Organization to INDEXED. Generally, the File name is the same as the Record name (unless you are using coded records).

Datatype Mappings

PowerHouse makes few restrictions on datatype usage for keys and indexes. File systems, on the other hand, are much more restrictive as to what is available, so PowerHouse maps its datatypes to the best match available when the file is generated using QUTIL. In some cases, a mapping that works well in IMAGE may not work well with C-ISAM or DISAM. You should review your key and index datatypes and adjust them where appropriate. A table of these datatype mappings can be found on (p. 60).

Automatic Masters

An IMAGE automatic master is a dataset containing a key only and no other items. IMAGE maintains the automatic master records, deleting and adding as needed. Automatic masters are typically used for two purposes. First, they can provide an easy way to ensure unique values by providing a lookup key. Second, they can provide a convenient retrieval path.

Since indexed files can provide the same functionality through unique indexes, you no longer need the record structures defined as IMAGE automatic masters. Instead, add a unique index to the new detail file. If you remove an automatic master, also replace any code that references the automatic master directly with code that references the unique index in the detail file.

An automatic master provides the key corresponding to the detail dataset search item. While IMAGE does not allow you to declare detail search items as unique indexes, PowerHouse does allow this, and automatically generates a LOOKUP NOTON. In cases such as these, no code changes are required.

On the other hand, if a FIELD statement for an item of the detail file does a LOOKUP NOTON the automatic master to ensure uniqueness, then the LOOKUP NOTON must be done against the detail file itself using the unique index.

Regarding search retrieval paths, in most cases the retrieval is done against the detail dataset in PowerHouse code, so removing the automatic master and replacing it with an index doesn't usually require any code changes.

Manual Masters and Details

Indexed file systems makes no distinction between files used for master or detail data. When converting your PDL, you just drop the Master or Detail option.

Similarly, the `LINKS TO record [SORT ON item]` option of the Index statement has no equivalent, so it can be dropped too. Just be sure to keep (or add) an index for each file.

Your QUICK and QTP programs may have special coding because IMAGE requires that (a) master data be added before corresponding detail data is added, and (b) detail data be deleted before corresponding master data is deleted. You should review this code to ensure that it is consistent with the new files and indexes.

QUICK Locking

When you use default locking in IMAGE, the PUT verb locks the dataset, does the update and unlocks the dataset. When you use default locking in C-ISAM or DISAM, the PUT verb locks just one data record at a time, updates it, and then unlocks it. If more than one record has been locked, completion of the PUT verb unlocks them all.

Miscellaneous Syntax

DBMODE syntax does not apply to indexed file systems, and must be removed. Depending on its purpose, it may need to be replaced with an access or exclusivity option.

- QUICK: `FILE x ... OPEN DBMODE n`
- QTP: `SET FILE X DBMODE n [DEFERRED]`
- QUIZ: `SET DBMODE n`

Indexed files cannot be opened with an access-type of Append or Clear (for example, "`FILE x OPEN APPEND`"). Valid types are Update (the default), Read, and Write.

In PDL, the following syntax does not apply to indexed file systems, and must be removed:

- `FILE x ... [NO]ITEM CRITICAL UPDATE`
- `RECORD x ... OPEN name INDEXED`
- `INDEX x ... LINK TO record [SORT ON item]`

In QUTIL, the following syntax does not apply to indexed file systems, and must be removed: `CREATE|DELETE BASE x`. In its place, either create/delete files individually (`CREATE FILE x`) or use `CREATE ALL`.

In QDESIGN, the following syntax does not apply to indexed file systems, and must be removed: `STARTLOG, STOPLOG, MEMOLOG`

DISAM Indexes

Windows stores numeric and character data in different internal formats ('little endian' and 'big endian' respectively). PowerHouse does these conversions automatically. However, there some resulting restrictions that you should be aware of. In particular, IMAGE indexes that use substructures should be changed to DISAM indexes that use segments.

For more information, see "DISAM File Support" in the *PowerHouse 4GL 8.4 New Features* document.

Migrating from KSAM (Indexed)

The most likely types of file systems you will migrate your KSAM files to are indexed or relational ones. The former is simpler and less expensive. The latter is more powerful but requires more migration work.

KSAM to Indexed

Both C-ISAM on UNIX and DISAM on Windows accept the vast majority of your PDL syntax for KSAM. Hence there are also few changes required to your application programs.

Overlapping Indexes

KSAM does not support multi-segment indexes. However PowerHouse allows you to declare multiple indexes starting in the same position, in which case QUTIL only generates the longest index. If you migrate this PDL structure to C-ISAM or DISAM, be sure to use QUTIL (rather than a C-ISAM or DISAM utility) to create the physical file. Otherwise the utility creates two separate indexes, and data retrieval through the shorter index may not be in the same order as it was with KSAM.

For example

```
> FILE KSAMEX ORG INDEXED TYPE KSAM
> RECORD KSAMEX
> ITEM K1
> ITEM K2
...
> INDEX KSAM-INDEX1
>   SEGMENT K1
>   SEGMENT K2
> INDEX KSAM-INDEX2
>   SEGMENT K1
```

Even though two indexes are declared, they both start in the same location and KSAM only allows one index to exist starting in that location. If QUTIL is used to create this file, it creates the single index. If data is retrieved using KSAM-INDEX2, the retrieval is done using a partial-key retrieval on the index that starts in the position of K1. However the sequence of retrieval is the same for KSAM-INDEX1 and KSAM-INDEX2.

C-ISAM and DISAM support multi-segment indexes and allow multiple indexes to have segments starting in the same position. In the previous example, the two indexes would be two physically separate indexes if the file was created by a C-ISAM or DISAM utility. In this case, retrieval using KSAM-INDEX2 will not necessarily be in the same sequence as KSAM-INDEX1 because the indexes are kept separately.

For C-ISAM and DISAM, QUTIL still creates a single index in situations like this, as long as the datatypes and positions are consistent. So if you want the same retrieval characteristics as you had with KSAM, it is recommended that you use QUTIL to generate the new C-ISAM or DISAM files from the dictionary definitions.

Datatype Mappings

PowerHouse makes few restrictions on datatype usage for keys and indexes. File systems, on the other hand, are much more restrictive as to what is available, so PowerHouse maps its datatypes to the best match available when the file is generated using QUTIL. In some cases, a mapping that works well in KSAM may not work well with C-ISAM or DISAM. You should review your key and index datatypes and adjust them where appropriate. A table of these datatype mappings can be found on (p. 60).

PDL

To change a PDL FILE statement to work with C-ISAM or DISAM, simply change the TYPE option from KSAM to CISAM (no hyphen) or DISAM. (On UNIX, the TYPE clause can optionally be omitted.) For example:

`FILE BILLINGS ORGANIZATION INDEXED` **`TYPE DISAM`** `...`

The following FILE options do not apply to C-ISAM or DISAM and must be removed:

- `ASCII|BINARY`
- `BLOCKING FACTOR n`
- `CAPACITY n`
- `KEYFILE x`
- `REUSE`

You may be able to remove duplicate record data. Since KSAM only allows one index that starts on a given column, you may have created a duplicate column to support two indexes with a common segment. Since C-ISAM and DISAM allow index segments that are not contiguous, you may want to remove the duplicate column and create the overlapping indexes.

QUICK

When you use default locking in KSAM, the PUT verb locks the file, does the update and unlocks the file. When you use default locking in C-ISAM or DISAM, the PUT verb locks just one data record at a time, updates it, and then unlocks it. If more than one record has been locked, completion of the PUT verb unlocks them all.

QTP

The following syntax does not apply to C-ISAM or DISAM and must be removed:

- `SET FILE x` **`LOCK|NOLOCK`**
- `SUBFILE ... INDEX x` **`ORDERED|UNORDERED`**

- `SUBFILE ... SIZE n`

QUIZ

The following syntax does not apply to C-ISAM or DISAM and must be removed:

- `SET LOCK|NOLOCK`
- `SET SUBFILE ... INDEX x ORDERED|UNORDERED`

DISAM Opens

Unlike KSAM or C-ISAM, a DISAM file opened for Write or Append cannot be read until it is closed. Review your programs to see if they are reading any KSAM files opened for Write or Append, and either issue a Close before the read or postpone the read.

With DISAM, Update Exclusive opens are not permitted.

DISAM Indexes

Windows stores numeric and character data in different internal formats ('little endian' and 'big endian' respectively). PowerHouse does these conversions automatically. However, there some resulting restrictions that you should be aware of. In particular, KSAM indexes that use substructures should be changed to DISAM indexes that use segments.

For more information, see "DISAM File Support" in the *PowerHouse 4GL 8.4 New Features* document.

KSAM to Relational

You can 'step up' to a relational database in order to get the various advantages these systems offer (such as set processing and structures that make data access cleaner and more flexible). Keep in mind that the structural changes can make for a more complex migration of your data, and many of the performance benefits will only be realized after making changes to your application programs.

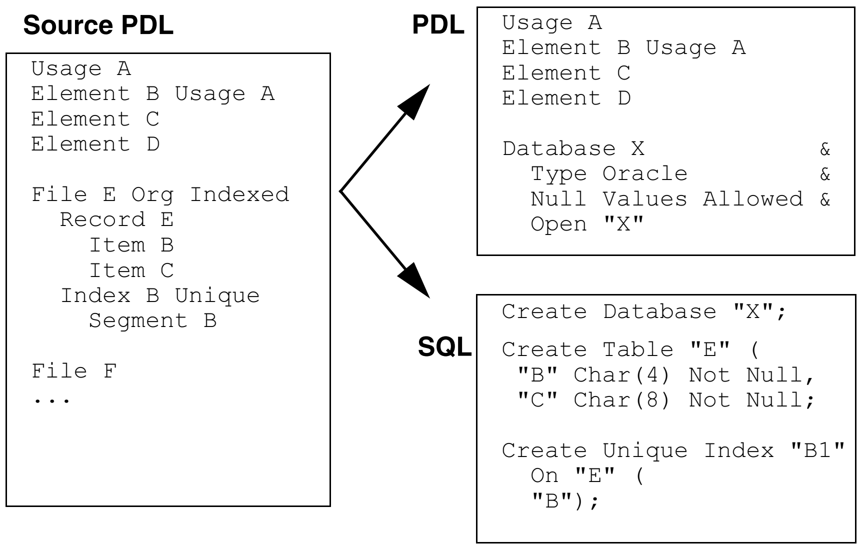
Cognos currently supports the following relational database management systems (RDBMSes).

UNIX	Windows
Oracle	Oracle
Sybase	Sybase
IBM DB2 UDB	IBM DB2 UDB
DataDirect ODBC to SQL Server	MS SQL Server 7 and 2000 (via ODBC)
ALLBASE/SQL (HP-UX only)	

You need to make several types of changes to your application source files, both data definitions and programs. This can be done manually, in an editor, or it can be done with automated tools. For information on these tools, see "The Migration Plan" (p. 63).

Data Definitions

Your PDL file for KSAM defined both logical objects (such as Elements) and physical storage objects (such as Files, Records, Items and Indexes). The PDL for a relational application contains only the logical objects and a *pointer* to the database. You also create a database SQL source file containing the SQL equivalent to your File, Record, Item and Index definitions.



```
Usage A
Element B Usage A
Element C
Element D

File E Org Indexed
  Record E
    Item B
    Item C
  Index B Unique
    Segment B

File F
...
```

```
Usage A
Element B Usage A
Element C
Element D

Database X           &
  Type Oracle        &
  Null Values Allowed &
  Open "X"
```

```
Create Database "X";

Create Table "E" (
 "B" Char(4) Not Null,
 "C" Char(8) Not Null;

Create Unique Index "B1"
  On "E" (
  "B");
```

The PDL Database statement often includes a database user ID and password within the Open Name.

In addition to Usage, Element, and Database statements, the PDL may also contain Application Security Class statements. PERMIT statements cannot be applied to relational data, but ASCs may still be used in the USERS INCLUDE options of screens, runs, and reports.

Use QSHOW to generate the initial SQL from your dictionary:

```
QSHOW
> SET LANGUAGE SQL
> FILE QSHOSQL=QSHO1
> GENERATE DATABASE E
> FILE QSHOSQL=QSHO2
> GENERATE DATABASE F
> FILE QSHOSQL=QSHO3
> GENERATE ...
```

This creates a series of text files, QSHOn, which you then merge and modify to include syntax specific to your chosen RDBMS. Alternatively, Axiant can convert your entire dictionary, as described in "The Migration Plan" (p. 63).

Identifying the Database

When accessing a relational table within a PowerHouse component, use the IN clause to identify the database as defined within your PDL. For example:

QDESIGN: `FILE ORDERS IN X`

QUIZ/QTP: `ACCESS ORDERS IN X`

Alternatives are the SET DATABASE statement (to provide the default database) and the program parameter SUBDICT=SEARCH. If all table names are unique, then using SUBDICT means not having to change source code.

Naming Conventions: Hyphens to Underscores

Relational databases do not permit hyphens ("-") within object names. This avoids confusion with the subtraction or minus character in operations.

The most common replacement character is an underscore ("_"). For example, "MONTH-NAME" becomes "MONTH_NAME". The change is a simple editor find-and-replace in your source files. However, do these one by one rather than with a wildcard, to avoid making the change in code performing a subtraction operation. Alternatively, the Axiant migration tools do this automatically.

Naming Conventions: Sybase Case Sensitivity

Sybase is case sensitive. However, PowerHouse upshifts object names by default. Hence, a Sybase table or column defined in mixed or lower case (for example, "Billings") would not be recognized by PowerHouse (where `ACCESS Billings` would become `ACCESS BILLINGS`).

There are several ways to avoid this issue. The simplest method is to disable the automatic upshifting and use the correct case in your PowerHouse programs. To disable the upshifting, add the following to your PDL:

```
SYSTEM OPTIONS SHIFT NOSHIFT
```

Normalization

RDBMSes achieve their benefits by working on the basis of normalized data. Normalization is a design process removing redundancy in your data definitions. Thus a migration to relational requires data structure changes while moving from PDL to SQL.

Ideally, you would make a full review of your record structures and implement a fully normalized data model. At a minimum, it is necessary to remove all non-relational features from your data definitions. This may result in additional data objects, but the resulting application programs will be easier to understand and maintain.

The following designs are not normalized:

- item substructures
- arrays (occurring or repeating items)
- item redefinitions
- multiple record layouts (used for coded records with Select values)

Item Substructures

Two ways to normalize substructures are

- remove the highest level item and use lower levels only

Unnormalized	**Normalized**

`Item MONTH-KEY`
`Begin Structure`
`  Item INITIALS`
`  Item MONTH`
`End Structure`
	`Item INITIALS`
	`Item MONTH`

- remove lower levels and use the highest level only

Unnormalized	**Normalized**

`Item TRANS-DATE`
`Begin Structure`
`  Item YEAR`
`  Item MONTH`
`  Item DAY`
`End Structure`
	`Item TRANS_DATE`

The first method is useful when the end user tends to enter the subitems individually; the second approach is useful when you commonly work with the whole structure at a time (for example, a date field).

If you choose the first method yet you need to access the higher-level item in your applications, you can create a concatenated item in either the application (for example, a Define) or the database (for example, a calculated column).

Similarly, if you choose the second method yet you need to access the lower-level item in your applications, you can extract it in either the application (for example, a Define) or the database (for example, a calculated column).

Arrays

Some RDBMSes support arrays, but they must be replaced with normalized structures for PowerHouse applications. Three ways to normalize arrays are

- define individual fields for each occurrence

Unnormalized Sales File	Normalized
`Item PRODUCT-NO`	`Item PRODUCT_NO`
`ITEM MONTHLY-TOTALS &`	`Item JANUARY_TOTAL`
`  Occurs 12`	`Item FEBRUARY_TOTAL`
`...`	`...`

- create a detail table (relational files are called tables)

Unnormalized Sales File	Normalized Tables
`Item PRODUCT-NO`	Sales table:
`ITEM MONTHLY-TOTALS &`	`Item PRODUCT_NO`
`  Occurs 12`	`...`
`...`	Monthly_Sales table:
	`Item PRODUCT_NO`
	`Item MONTH`
	`Item MONTHLY_TOTAL`

- define a single item as long as all the occurrences

  This solution is only available if the items are character or zoned unsigned. For example, `Item DAY-CODE Char Size 2 Occurs 7` could become `Item DAY_CODE Char Size 14`. You would then use the [n:m] function in your applications to extract the desired day.

The first method is the simplest metadata change, but it is less practical if there are a large number of occurrences, and it is not true normalization. It can result in many lines of application code. For example, a screen cluster around one Field statement would be replaced by multiple Field statements; a procedural For loop would be replaced similarly. This method might be appropriate for MONTH (12 values) but not for DAY_OF_MONTH (31 values).

The second method may be less efficient when reading, as the detail table holds one record for each occurrence. However it can be more efficient in program processing logic, and require simpler program changes. The I/O loss is often recovered by both the CPU savings and the easier maintenance effort. As well, aggregate processing can be simpler and faster if you change your logic to use SQL.

Screen Example, Unnormalized Sales File	Screen Example, Normalized Tables
`Screen X`	`Screen X`
`File SALES`	`File SALES` **`In DB`**
	`File MONTHLY_SALES In DB &`
	`  Detail Occurs 12`
`Field PRODUCT_NO`	`Field PRODUCT_NO`
`Cluster Occurs with &`	`Cluster Occurs with &`
`  MONTHLY_TOTAL`	**`  MONTHLY_SALES`**
`Field MONTHLY_TOTAL`	`Field MONTHLY_TOTAL`
`End Cluster`	`End Cluster`
`...`	`...`

Item Redefinitions

Two ways to normalize Redefines are

- to remove one item and change program logic to only use that one

Unnormalized	**Normalized**

`Item USERID`
`Item USERIDCHAR &`
`  Redefines USERID`
	`Item USERID`

- to keep separate items and maintain them in your program logic

Unnormalized	**Normalized**

`Item USERID`
`Item USERIDCHAR &`
`  Redefines USERID`
	`Item USERID`
	`Item USERIDCHAR`

Which method you use depends on the reason for the original Redefine. The second method need not require changes to your application code if you use RDBMS triggers or computed fields to automatically populate one item with the value of the other item.

Multiple Record Layouts

Multiple record layouts for a single file are usually (but not necessarily) used when creating coded records with Select clauses. For example:

```
File CONTACTS
  Record CONTACTS
    Item RECTYPE
    ...
  Record SUPPLIERS
    Item RECTYPE SELECT "S"
    ...
```

Two ways to normalize multiple record layouts are:

- use one table and add views to separate the data
- create separate tables and merge them with a view

If you use a single large table, you can create views that report from specific subsets of that data. Depending on the RDBMS and the Select statement within the view, you may also be able to write to the table through the view.

If you choose to use separate tables, you may be able to report from all tables at once by creating a database view that includes the data from all the tables.

Program changes are likely necessary after removing coded records. This includes changing or removing Select options and "Set File Open n" statements.

Primary Keys and Indexes

A primary key is a unique identifier of each record in the table. Some RDBMSes, such as Sybase, require a primary key in every table definition. Regardless of the RDBMS, we recommend you add a primary key and a unique index to every table if they do not already have one. This maintains data integrity. Equally important, it allows table records to be updated or deleted within Cursors. Cursors are a relational feature which can greatly improve performance.

Primary keys and indexes should be modified if they contain an item which was a substructure and you have replaced the substructure with the lower-level items (p. 42). The key or index must have its segments defined with the same items.

Some RDBMSes, such as Oracle and DB2, automatically generate a unique index based on the primary key for every table. You may want to disable this database feature if you have your own unique indexes already declared. (An alternative is to remove your own index definitions, but you then have to also change any programs that refer to those indexes through syntax like VIAINDEX.)

Index names must be unique within a relational database, so you may have to rename some indexes with are currently the same within different record structures. Don’t forget to change any programs that reference the old index names.

Ordered Retrieval

Records in a relational table are not stored in a specific order (for example, chronological or by unique index). Therefore any programs that expect the data in a specific order but which have no Sort statement need to be changed.

In the following examples, the new code is shown in bold:

QUICK

```
FILE EMPLOYEES IN TRACKING
ACCESS ORDERED &
  VIA LAST_NAME USING LAST_NAME REQUEST LAST_NAME
```

QUIZ/QTP

```
ACCESS EMPLOYEES IN TRACKING
CHOOSE VIAINDEX LAST_NAME
```

For more information on ordering, see the Ordered, Orderby and Viaindex clauses in the *QDESIGN, QUIZ,* and *QTP Reference* documents.

Null Values

Relational databases support the concept of null values by default. As this affects programming logic and thus requires evaluation of all your program code, you may wish to disable null support initially and investigate its benefits after your system has been fully migrated.

A null value is an unknown value. It indicates nothing, not zero or blank. Nulls model the real world better than binary processing (True or False) does, because the real world often has missing information.

For example, calculations containing null values evaluate to null. So Total = Supply + Labour evaluates to null if either Supply or Labour is null. If you don't know what the labour costs are, you don't know what the total cost is. (Compare this to binary logic, where Total = Supply if Labour has nothing in it, because 'nothing' would be treated as a zero.)

Null support can cause conversion issues when loading your MPE/iX data into a relational database. It can also cause logic errors in existing program code that isn't expecting null processing rules. For more information on null values in a PowerHouse context, see the *PowerHouse Rules* document, as well as the training courses, *The PowerHouse Relational Interface (Parts I and II)*.

Null value processing is disabled within PowerHouse by default: the PDL Database statement has the default option of 'Null Values Not Allowed'. This can be changed once the migration is complete and you are ready to upgrade your programs to support nulls.

Miscellaneous Syntax

Relational tables cannot be opened with an access-type of Append, Clear, or Write (for example, `FILE x IN db OPEN APPEND`). Valid types are Update (the default) and Read.

Relational tables can only be opened with an exclusivity of Share. We recommend you replace exclusivity control with transaction control. For more information, see "The Next Step: Relational Features" (p. 57).

The QDESIGN Lock and Unlock verbs, as well as the Lock option are generally not supported for relational tables. The same is true for the QTP `SET FILE [UN]LOCK` and QUIZ `SET [UN]LOCK` statements. Allbase and ORACLE have limited support for them, and they are ignored for other database systems. In any case, for all RDBMSes, we recommend they be replaced with transaction control statements. For more information, see "The Next Step: Relational Features" (p. 57).

The creation and maintenance of your relational database is handled by database tools rather than PowerHouse. Thus, in QUTIL, the following syntax does not apply to relational tables, and must be removed: `CREATE|DELETE FILE x`

Two PowerHouse options which behave differently in a relational application are Open numbers (for example, "FILE x Open 1") and the Close verb or option.

- Open numbers cause separate transactions to be created. We recommend they be replaced with transaction control statements.

- The Close verb or option commits a transaction and then detaches from the database. We recommend they be replaced with a Commit verb.
- For more information, see "The Next Step: Relational Features" (p. 57).

The Next Step: Relational Features

Once you have successfully migrated your application to a relational environment, your next step is to take advantage of features that were not available with KSAM.

Some of these features can simplify your programs, some can extend your program functionality, and several of them can improve performance. Some of these features require changes to the database, while others require changes to your program code.

Some features to consider adding to the database are:

- triggers (SQL code inserted in the database to take place whenever any application inserts, modifies, or deletes data in a specified table);
- views (virtual tables, constructed in the database to merge or separate data from one or more tables)
- computed fields (columns whose definition includes a calculation, performed at write time once, which improves read time);
- varchars (variable character datatypes, which can be more efficient than fixed-length character items).

Some features to consider adding to your programs are:

- SQL cursors, essentially temporary views defined for the duration of that program run;
- SQL commands (Insert, Update, and Delete) that you can include within QUICK procedures and QTP calls;
- transaction control, the relational method of controlling data access;
- null value support, allowing your program logic to handle real-world situations of missing data;
- stored procedures, functions defined within the database which can be explicitly called by QUICK or QTP.

For more information, see the *PowerHouse and Relational Databases Reference* document and the training courses, *The PowerHouse Relational Interface (Parts I and II)*.

Migrating from MPE Files (Direct, Relative, Sequential)

A sequential file is one which can be read sequentially. New records are added to the end of the file. Existing records can't be updated or deleted.

A direct file is one which can be read sequentially or by record number. New records are added to the end of the file. Existing records can be updated but not deleted.

A relative file is like a direct file except that records can also be deleted (leaving gaps in the file).

Direct and Sequential Files

UNIXIO on UNIX supports direct and sequential files. The same is true for DOS on Windows. Hence there are also few changes required to your application programs.

PDL

To migrate a Direct or Sequential PDL FILE statement, simply change the TYPE option (or remove it completely, since it is optional):

MPE/iX: `FILE x ORGANIZATION DIRECT|SEQUENTIAL` **`TYPE MPE`**

UNIX: `FILE x ORGANIZATION DIRECT|SEQUENTIAL` **`TYPE UNIXIO`**

Windows: `FILE x ORGANIZATION DIRECT|SEQUENTIAL` **`TYPE DOS`**

The following FILE options do not apply to UNIXIO or DOS and must be removed:

- ASCII|BINARY
- BLOCKING FACTOR n
- CAPACITY n

QUICK

When you use default locking with MPE files, the PUT verb locks the file, does the update and unlocks the file. When you use default locking with Direct or Sequential files on UNIX and Windows, the PUT verb locks just one data record at a time, updates it, and then unlocks it.

QTP

The "`SET FILE x LOCK|NOLOCK`" syntax does not apply to UNIXIO or DOS files and must be removed.

QUIZ

The "`SET LOCK|NOLOCK`" syntax does not apply to UNIXIO or DOS files and must be removed.

Relative Files

Unix and Windows do not support relative files (which are like direct files that also allow record deletion). To support the deletion capability, we recommend such files be migrated to an indexed file system (C-ISAM on UNIX or DISAM on Windows).

This can be done with a simple index consisting of a one-segment sequence number.

On MPE/iX:

```
FILE x ORGANIZATION RELATIVE TYPE MPE
  RECORD x
    ITEM I1...
    ITEM I2...
```

On UNIX and Windows:

```
FILE x ORGANIZATION INDEXED TYPE CISAM|DISAM
  RECORD x
    ITEM I1...
    ITEM I2...
    ITEM SEQUENCE_NO
  INDEX x1 UNIQUE
    SEGMENT SEQUENCE_NO
```

You must modify any code that writes to the file to also populate the index value. For example, it could read the last record for its sequence number and then add that value plus one to the new record. You may need to apply some locking in a concurrent-user environment.

Retrieval must be changed from a direct read (USING n) to an indexed retrieval (VIA segment USING n).

PDL

The following FILE options do not apply to UNIXIO or DOS and must be removed:

- `ASCII|BINARY`
- `BLOCKING FACTOR n`
- `CAPACITY n`

QUICK

When you use default locking with MPE files, the PUT verb locks the file, does the update and unlocks the file. When you use default locking with Indexed files, the PUT verb locks just one data record at a time, updates it, and then unlocks it. If more than one record has been locked, completion of the PUT verb unlocks them all.

QTP

The "`SET FILE x LOCK|NOLOCK`" syntax does not apply to C-ISAM or DISAM files and must be removed.

QUIZ

The "`SET LOCK|NOLOCK`" syntax does not apply to C-ISAM or DISAM and must be removed.

Migrating from Allbase or Oracle (Relational)

We recommend that MPE/iX Allbase and Oracle databases be migrated to Allbase (on HP-UX) or Oracle (on UNIX or Windows). However a migration to any other RDBMS would also be a viable option.

Study the documentation on your chosen RDBMS and compare it to your source RDBMS to identify the changes you need to make to your SQL data definition file. Common issues are the syntax structure, functions, and datatypes supported. Depending on your data definition changes, this may also require modifications to your PowerHouse program logic.

Cognos currently supports the following relational database management systems (RDBMSes).

UNIX	**Windows**
Oracle	Oracle
Sybase	Sybase
IBM DB2 UDB	IBM DB2 UDB
DataDirect ODBC to SQL Server	MS SQL Server 7 and 2000 (via ODBC)
ALLBASE/SQL (HP-UX only)	

Naming Conventions: Sybase Case Sensitivity

Sybase is case sensitive, however PowerHouse upshifts object names by default. Hence, a Sybase table or column defined in mixed or lower case (for example, "Billings") would not be recognized by PowerHouse (as "ACCESS Billings" would become "ACCESS BILLINGS").

There are several ways to avoid this issue. The simplest method is to disable the automatic upshifting and use the correct case in your PowerHouse programs. To disable the upshifting, add the following to your PDL:

```
SYSTEM OPTIONS SHIFT NOSHIFT
```

Primary Keys and Indexes

A primary key is a unique identifier of each record in the table. Some RDBMSes, such as Sybase, require a primary key in every table definition. Regardless of the RDBMS, we recommend you add a primary key and index to every table if they do not already have one. This maintains data integrity. Equally important, it allows table records to be updated or deleted within Cursors, a relational feature which can greatly improve performance.

Some RDBMSes, such as Oracle and DB2, automatically generate a unique index based on the primary key for every table. You may want to disable this database feature if you have your own unique indexes already declared. (An alternative is to remove your own index definitions, but you then have to also change any programs that refer to those indexes through syntax like VIAINDEX.)

After the Migration

Once your migration is completed, study the documentation of the new RDBMS to identify features that were not present on the source RDBMS. You may want to take advantage of some of these by further refining your SQL and programs.

Datatype Mappings

PowerHouse makes few restrictions on datatype usage for keys and indexes. File systems, on the other hand, are much more restrictive as to what's available, so PowerHouse maps its datatypes to the best match available when the file is generated using QUTIL. In some cases, a mapping that works well in IMAGE or KSAM may not work well with C-ISAM or DISAM.

If you are migrating from IMAGE or KSAM to C-ISAM or DISAM, you should review your key and index datatypes using the following table and adjust them where appropriate. If you create the files outside of PowerHouse, ensure that the indexes you create are consistent with the PowerHouse definitions.

Since PowerHouse does not generate relational databases using QUTIL, it makes no assumptions about relational datatypes.

PowerHouse Datatype	**IMAGE Datatype**	**KSAM Datatype**	**C-ISAM/DISAM Datatype**
CHARACTER	X	BYTE	CHARTYPE
DATE	I2	BYTE	CHARTYPE
DATETIME	I2	BYTE	CHARTYPE
FLOAT	E2 (4 byte) E4 (8 byte)	REAL (4 byte) REAL (8 byte)	FLOATTYPE (4 byte) DOUBLETYPE (8 byte)
FREEFORM[1]	X	BYTE	CHARTYPE
INTEGER SIGNED[2]	I	INTEGER	INTTYPE (2 bytes) LONGTYPE (4 bytes) CHARTYPE (> 4 bytes)
INTEGER UNSIGNED	K	BYTE	CHARTYPE
INTERVAL	R4	LONG	DOUBLETYPE
JDATE	K	BYTE	CHARTYPE
PACKED SIGNED[3]	P	PACKED	CHARTYPE
PACKED UNSIGNED[3]	P	PACKED	CHARTYPE
PHDATE	K	BYTE	CHARTYPE
VARCHAR[4]	--	BYTE	CHARTYPE
ZDATE	X6	BYTE	CHARTYPE
ZONED SIGNED[3]	Z	BYTE	CHARTYPE
ZONED UNSIGNED[3]	Z	NUMERIC	CHARTYPE

PowerHouse Datatype	IMAGE Datatype	KSAM Datatype	C-ISAM/DISAM Datatype

[1] FREEFORM keys and segments do not work due to the nature of the FREEFORM datatype.

[2] CHARTYPE won't handle negative numbers correctly.

[3] PACKED and ZONED only work for C-ISAM and DISAM if the signs are all the same.

[4] The first two bytes which represent the size are ignored.

Moving Forward

"The Migration Plan" (p. 63) describes the 12-step plan to actually perform the migration. This identifies everything from identifying training needs to editing data definitions.

Chapter 5: The Migration Plan

Overview

At this stage, you've evaluated the choices for both your target platform and your target file system(s). This chapter describes the twelve step plan to actually perform the migration. This covers everything from identifying training needs to editing data definitions, with an emphasis on the physical migration activities in Step 6.

The Twelve Step Plan

1. Define the intended application.
2. Learn about the target environment and migration tools.
3. Take inventory of what you have.
4. Identify what will be migrated.
5. Develop a plan.
6. Migrate the pilot application.
7. Assemble the pilot application.
8. Assess the results of Steps 2 through 7.
9. Have users evaluate the pilot application.
10. Assess the results of Steps 2 through 9.
11. Plan the complete migration.
12. Do the complete migration.

Step 1: Define the Application

This simple step has you document exactly how much of the source application will be migrated to the target platform and file system(s). Identify the architectures you have to create to support the new environment. Also define how you will measure success, in terms of user acceptance and business productivity.

An important component of this step is to identify a portion of the application that you will initially migrate as a **pilot**. This can be used to evaluate your tools, processes, and understanding of the migration issues.

As such, choose a portion which is small but which contains a variety of features, such as:

- all PowerHouse components
- calls to operating system commands
- 3GL calls (if the application has many)
- data definitions that have changed significantly (for example, due to a change of the file system and supported features)
- security

Step 2: Learn

Learn about your target platform and file system(s). If you will be using the Axiant migration tools in step 6, learn about those as well.

This learning can be done by reading the appropriate documentation, discussing the issues with knowledgeable co-workers, or taking training courses. The time and costs invested in education will be recovered later by a smoother migration.

Cognos courses to consider include

- *Migrating Applications with Axiant 4GL*
- *The PowerHouse Relational Interface (Parts I and II)*
- *Axiant 4GL - Building Applications*
- *PowerHouse Web - Developing Applications*

For information on these and other courses, see http://support.cognos.com/.

Step 3: Take Inventory

Take an inventory on both your MPE/iX system and your target system to identify

- skills

  Who knows what about PowerHouse, its relational interface, Axiant, the source and target operating systems and file systems, the networking, the user interface, and the applications involved? This may involve existing staff, new staff, and external consulting services. All team members should be familiar with the key architectural features of the various technologies in order to eliminate potential problems early in the process. We also recommend including an application user on the migration team to get early buy-in and catch user issues at an early stage.
- software

  Identify what software is needed on both the source and the target operating systems and file systems. This includes PowerHouse versions, editors, mail systems, possible 3GLs, batch processing tools, security systems, and FTP tools. Axiant includes a Migration Profile tool that identifies the programs and data used, showing them in a graphical format.
- hardware

  What are the hardware requirements for your Windows or UNIX servers (for example, disk space, memory, and networks)?
- standards and practices

  Identify day-to-day activities that may have to change due to the new environment. For example, perhaps users were notified of software updates indirectly by having the system go down during changes. If software updates could now be done automatically over a network, you have to specifically plan to notify the users.

Step 4: Identify What Will Be Migrated

Determine what will be

- moved as is

  Many screens, runs, and reports will not have to change, especially if the source and target file systems are similar (for example, IMAGE to Eloquence, or KSAM to C-ISAM or DISAM).
- moved with changes

  For example, a move to an RDBMS will require replacing substructures, arrays, and redefinitions. These, in turn, will lead to changes in your programs. Any platform-specific logic, such as UDCs, also falls into this category. A 3GL program may need changing due to different compilers. A migration to an RDBMS will allow optional program changes to use cursors and other SQL syntax.
- replaced

  IMAGE databases may be replaced with new relational databases or indexed files. QUIZ reports may be replaced by GUI tools such as ReportNet, Impromptu, and PowerPlay.
- created

You may have to create some new tables to make up for non-relational structures in IMAGE or KSAM files. You may also want to create new relational database objects (such as views and triggers), which in turn will affect your programs.

Step 5: Develop a Plan

A plan for the move will include a preliminary schedule. This should include time for:

- training
- design
- development
- help systems
- the actual migration
- testing (security, for example, and running the new application in parallel with the source application)

We recommend that you meet with your Cognos representatives during the development of the migration plan. They have lots of migration experience and may also be aware of recently available tools or tips. If you are moving to a new file or database system, you should include the vendor in the planning as well.

Through consulting programs and services, Cognos and other vendors can provide skilled resources, either directly or through a partner, to help fill temporary personnel or expertise needs. For more information, see http://powerhouse.cognos.com/.

Step 6: Migrate The Pilot Application

The pilot portion of the overall application was identified in Step 1. The actual migration activities will vary, depending on whether you are using the Axiant migration tools.

- "Migrate with the Axiant Migration Tools" (p. 65)
- "Migrate without the Axiant Migration Tools" (p. 68)

For information on this decision, see (p. 13).

Migrate with the Axiant Migration Tools

If you are moving to UNIX or Windows and employing the Axiant migration tools on Windows as a middle step, your migration will involve the following activities:

1. Collect your MPE/iX files (as identified in Steps 3 and 4).
2. Send your files to Windows.
3. For the pilot portion of your application, convert your files: add file extensions, then use the Axiant migration tools to convert the PDL and component programs to formats for your target platform and file system(s).
4. Send your modified PDL and component programs to your target platform (UNIX or the appropriate Windows server).

1. Collect your MPE/iX Files

Obtain a current PDL definition of your MPE/iX application, or use QSHOW to generate one:

```
> SET LANGUAGE PDL
> SET SECURITY
> GENERATE ALL
```

Use QTP to transfer the data into portable subfiles. Also convert any data currently in permanent subfiles to portable subfiles for transfer to the new platform. Use QTP instead of QUIZ because of internal differences in generating the minidictionaries (such as the way arrays are described). QUIZ subfile contents are also based on the items in the REPORT statement, which has a maximum record length of 256 characters.

When writing your data to portable subfiles, you will not usually have to code ITEM statements. For example:

QTP:

```
ACCESS PAY
SUBFILE PAYSF KEEP PORTABLE INCLUDE PAY
GO
```

If you are migrating from IMAGE to Eloquence, QUTIL can generate the equivalent Eloquence databases, or you can use the IMAGE schema. Eloquence provides a tool for moving the IMAGE data directly.

2. Send Files to Windows

On MPE/iX, we suggest you place your various application files into separate groups based on component (for example: PDL, QUICK, QTP, QUIZ, SUBFILES, 3GLS, and UDCS). They can then be sent to separate corresponding folders on Windows. Use an FTP tool to transfer all of your application files from MPE/iX to the corresponding folders in the Windows environment.

When using FTP, recall that each portable subfile is made up of a data dictionary file (such as SUB1) and a data file (such as SUB1Q). Use the ASCII file transfer format when sending all files from MPE/iX to Windows, ***except*** for the data file of each subfile.

It is important to send the data files for all subfiles using the BINARY transfer format of the FTP tool (even though the data files simply contain ASCII text). If you use the ASCII transfer format, it will insert a line feed character at the end of each record. PowerHouse subfile data files should not have this character inserted.

3. Convert your Files

Within a Command Prompt window, rename your files to have PowerHouse extensions (lower case for UNIX, either case for Windows). This is easiest if you have moved your files into separate folders based on the component during the FTP operation. For example:

- Move into the SUBFILES folder, then type "`rename * *.psd`" followed by "`rename *Q *.ps`" to give the subfiles their extensions.
- Use similar commands in the other folders (such as "`rename * *.qks`" for QUICK) to give the other files their extensions.

Within Axiant:

1. Create a Migration Profile defining your source and target platforms and file systems (for example, MPE/iX and IMAGE to UNIX and relational).
2. Import the data definitions:
   - Use the Migration Profile to import your PDL and generate the dictionary objects (such as Elements and Files).

     If you are migrating to an RDBMS, this also generates database objects (such as tables) and makes automatic global changes to normalize the data (removing arrays, substructures, item redefinitions and coded records).
   - Apply your own global changes to your dictionary and database objects, using the Change Manager. This can include name changes (such as hyphens to underscores), key and index changes, and customized solutions for normalization.
3. Import the component programs:
   - Use the Migration Profile to generate Axiant program objects (such as screens, runs, and reports) from your program source files.

     You can group programs into program sets to identify the work related to only your pilot application.

     The Advice tool identifies migration issues to be aware of (such as MPE/iX-specific code).

     Name changes applied during the data definition migration are automatically applied to your programs. You can also customize any program changes for the new platform or file system.
4. Export the data definition source files:

- Use the Generate PDL tool to create the PDL file for your target platform and file system, automatically removing options specific to MPE/iX and its file systems.
- Use the Maintain Databases tool to create the SQL file if you are migrating to an RDBMS.

  You can now edit these files for further customization, if desired.

5. Export the component program source files:
   - When you imported the MPE/iX program source files into Axiant, they were converted into Axiant objects. Compiling the application with a UNIX or Windows Build Profile creates *.qks, *.qts, *.qzs and other source files for your platform, automatically removing options specific to MPE/iX and its file systems. Programs for a relational file system also have the "IN database" option added.
   - You can now edit these files for further customization. (For example, QDESIGN procedural code statements with syntax specific to MPE/iX are not automatically edited, so this is a good time to make these changes.)
   - Use the Build Profile tool to identify what you want to compile, along with any compile options.

     You can specify a remote compile directly across the network to your target UNIX or Windows server, or you can do a local compile on your PC and save the resulting source files, to be sent to your target platform later (using a tool like FTP for UNIX or a network copy for Windows). The local compile approach allows you to separate the compile and send processes. However you must remember to set the Keep Temporary Files flag in your Build Profile.

Axiant includes a powerful Find Object tool for making global changes to programs while they are still Axiant objects. However you may prefer to make your edits after generating the source files, using a Windows or UNIX text editor. This will depend on your own comfort level with the various tools.

For target file systems like Oracle and DB2, you may wish to disable the generation of primary keys. Oracle and DB2 automatically create a unique index for each primary key, but they only allow one index on the primary key and you likely have your own index defined. To avoid this conflict in Step 7, clear the "Generate: primary keys" check box in the Migration Profile during your data definitions import.

For a target file system that is relational, Axiant also generates foreign keys when it detects a table whose primary key is a column in another table. Foreign keys, similar to the LOOKUP ON syntax in QDESIGN but evaluated at write time, are a useful component of referential integrity within a relational database.

An important edit task still remains, removing the 'Q' from the end of the filename for the data portion of all subfiles (*Q.ps). And for a target file system that is relational, the subfile dictionaries (*.psd) also need to have all hyphens replaced with underscores (except for Leading or Trailing minus signs). This two tasks can be easily done with UNIX scripts, so they will be discussed in "4. Send Files to Your Target Platform" (p. 67).

4. Send Files to Your Target Platform

If your target platform is a Windows server, simply copy all the files for your pilot application to that server, ideally preserving the separate folder structures.

If your target platform is a UNIX server, follow the same FTP procedure used when sending the original files from MPE/iX to Windows, ideally preserving the separate folder structures. (If you are using remote compiles, you won't need to resend the program source files.) Again ensure that all files are sent with the ASCII transfer format ***except*** for the data file of each portable subfile (which uses the BINARY transfer format).

You can now use a simple UNIX script to remove the 'Q' from the end of the filename for the data portion of all subfiles (*Q.ps). This is necessary because PowerHouse on UNIX and Windows uses file extensions instead of the 'Q' to distinguish the data file from the dictionary file.

- In an editor, create the script remove_q.txt in the SUBFILES directory:
  ```
  foreach p (`ls *Q.ps | sed 's/Q\.ps//g'`
    mv ${p}Q.ps $p.ps
  end
  ```
  Note: the sed string (`'s/Q\.ps//g'`) is placed within single quotation marks. But the outer foreach string (`` `...` ``) is placed within grave (pronounced 'grahv') marks. The grave key is located above the Tab key on most full sized keyboards.
- On UNIX, type `source remove_q.txt` to invoke the script. On Windows, this can be done after installing a UNIX emulation toolkit (such as MKS).

For a target file system that is relational, the subfile dictionaries (*.psd) also need to have all hyphens replaced with underscores (except for Leading or Trailing minus signs).

- In an editor, create the script sub_strings.txt in the SUBFILES directory:
  ```
  s/-/_g
  s/"_/"-/g
  ```
- In an editor, create the script replace_hyphens.txt in the SUBFILES directory:
  ```
  foreach p (`ls *.psd`)
    sed -f sub_strings.txt $p > tempfile
    mv tempfile $p
  end
  ```
  Note: the ls command (`` ls `...` ``) is placed within grave (pronounced 'grahv') marks, not quotation marks. The grave key is located above the Tab key on most full sized keyboards.
- On UNIX, type `source replace_hyphens.txt` to invoke the script. On Windows, this can be done after installing a UNIX emulation toolkit (such as MKS).

Migrate without the Axiant Migration Tools

If you are not employing the Axiant migration tools, your migration will involve the following activities:

1. Collect your MPE/iX files (as identified in Steps 3 and 4 of the Twelve Step Plan).
2. Send your files to your target platform.
3. For the pilot portion of your application, convert the PDL and component programs to formats for your target platform and file system(s).

1. Collect your MPE/iX Files

Obtain a current PDL definition of your MPE/iX application, or use QSHOW to generate one:

```
> SET LANGUAGE PDL
> SET SECURITY
> GENERATE ALL
```

Use QTP to transfer the data into portable subfiles. Also convert any data currently in permanent subfiles to portable subfiles for transfer to the new platform. Use QTP instead of QUIZ because of internal differences in generating the minidictionaries (such as the way arrays are described). QUIZ subfile contents are also based on the items in the REPORT statement, which has a maximum record length of 256 characters.

When writing your data to portable subfiles, you will not usually have to code ITEM statements. For example:

If you are migrating from IMAGE or KSAM to an RDBMS, also run QSHOW to generate an initial SQL definition from your dictionary. (See (p. 41).)

If you are migrating from IMAGE to Eloquence, QUTIL can generate the equivalent Eloquence databases, or you can use the IMAGE schema. Eloquence provides a tool for moving the IMAGE data directly.

2. Send Files to Target Platform

On MPE/iX, we suggest you place your various application files into separate groups based on component (for example: PDL, QUICK, QTP, QUIZ, subfiles, 3GLs, and UDCs). They can then be sent to separate corresponding folders on Windows. Use an FTP tool to transfer all of your application files from MPE/iX to the corresponding folders in the Windows environment.

When using FTP, recall that each portable subfile is made up of a data dictionary file (such as SUB1) and a data file (such as SUB1Q). Use the ASCII file transfer format when sending all files from MPE/iX to UNIX, ***except*** for the data file of each subfile.

It is important to send the data files for all subfiles using the BINARY transfer format of the FTP tool (even though the data files simply contain ASCII text). If you use the ASCII transfer format, it will insert a line feed character at the end of each record. PowerHouse subfile data files should not have this character inserted.

3. Convert Your Files

This section only applies to the pilot portion of your application.

First, rename your files to have PowerHouse extensions (lower case for UNIX, either case for Windows). This is easiest if you have moved your files into separate folders based on the component during the FTP operation. For example

❑ on UNIX

- Move into the SUBFILES folder, then type "`mv * *.psd`" followed by "`mv *Q *.ps`" to give the subfiles their extensions.
- Use similar `mv` commands in the other folders (such as "`mv * *.qks`" for QUICK) to give the other files their extensions.

❑ on Windows (within a Command Prompt window)

- Move into the SUBFILES folder, then type "`rename * *.psd`" followed by "`rename *Q *.ps`" to give the subfiles their extensions.
- Use similar `rename` commands in the other folders (such as "`rename * *.qks`" for QUICK) to give the other files their extensions.

Second, optionally use some unsupported UNIX tools to identify code that needs changing and to automatically make some of those changes. (If your target platform is Windows, you can still use the first two of these tools if you have software emulating the UNIX environment on Windows, such as the MKS Toolkit.) The tools are known as PowerHouse Migration scripts, and they can be downloaded from http://support.cognos.com/, under Documents, PowerHouse 8.4, and Utility.Run some or all of them, depending on whether they apply to your goals:

- **`phflag`**: This script searches program source files for MPE/iX-specific syntax. An asterisk is inserted in the first column of each line containing non-portable syntax, and the program is placed in the directory specified by the PHFIX environment variable.
- **`phparse`**: This script parses program source files and moves programs that fail to compile into the directory specified by the PHFIX environment variable.
- **`phlower`**: This script downshifts all characters in a set of files except for quoted strings and comments after a semicolon. Use this if you are renaming all your data objects and file names to lower case. **Note**: before executing this script, you must download and compile the C program **`phdown`**, which is called from **`phlower`**.
- **`phfilename.csh`** or **`phfilename.sh`**: This script alters the case of file names. Use this, along with phlower, if you are renaming all your file names to lower case.

Third, edit your PDL to adapt it to your new platform and file system(s). Use the results of the phflag and phparse scripts, as well as the appropriate earlier chapters in this guide:

- "The Target Platform: UNIX" (p. 15)
- "The Target Platform: Windows" (p. 27)
- "IMAGE to An IMAGE Emulator" (p. 39)
- "IMAGE to Relational" (p. 40)
- "IMAGE to Indexed" (p. 48)
- "KSAM to Relational" (p. 51)

- "KSAM to Indexed" (p. 49)
- "Migrating from Allbase or Oracle (Relational)" (p. 59)
- "Direct and Sequential Files" (p. 57)
- "Relative Files" (p. 58)

If you are migrating to an RDBMS, you must also edit or create an SQL data definition file, as described in "The File Systems" (p. 39).

If you are migrating to an IMAGE emulator, you must transfer your data definitions into the emulator database, following the steps indicated by the product supplier.

Fourth, edit your component programs to adapt them to your new platform and file system(s). As with your data definition changes, this can be done using the results of the phflag and phparse scripts as well as the appropriate earlier chapters in this guide.

You can now use a simple UNIX script to remove the Q from the end of the filename for the data portion of all subfiles (`*Q.ps`). This is necessary because PowerHouse on UNIX and Windows uses file extensions instead of the Q to distinguish the data file from the dictionary file.

- In an editor, create the script remove_q.txt in the SUBFILES directory:
  ```
  foreach p (`ls *Q.ps | sed 's/Q\.ps//g'`
    mv ${p}Q.ps $p.ps
  end
  ```
  Note: the sed string (`'s/Q\.ps//g'`) is placed within single quotation marks. But the outer foreach string (`` `...` ``) is placed within grave (pronounced 'grahv') marks. The grave key is located above the Tab key on most full sized keyboards.
- On UNIX, type `source remove_q.txt` to invoke the script. On Windows, this can be done after installing a UNIX emulation toolkit (such as MKS).

For a target file system that is relational, the subfile dictionaries (*.psd) also need to have all hyphens replaced with underscores (except for Leading or Trailing minus signs).

- In an editor, create the script sub_strings.txt in the SUBFILES directory:
  ```
  s/-/_g
  s/"_/"-/g
  ```
- In an editor, create the script replace_hyphens.txt in the SUBFILES directory:
  ```
  foreach p (`ls *.psd`)
    sed -f sub_strings.txt $p > tempfile
    mv tempfile $p
  end
  ```
  Note: the ls command (`` ls `...` ``) is placed within grave (pronounced 'grahv') marks, not quotation marks. The grave key is located above the Tab key on most full sized keyboards.
- On UNIX, type `source replace_hyphens.txt` to invoke the script. On Windows, this can be done after installing a UNIX emulation toolkit (such as MKS).

Step 7: Assemble the Pilot Application

At this stage, you have all the source files (both data definitions and programs) on your target platform and converted to work with your target file system(s). To assemble the pilot application:

- ❑ Build the relational database (if applicable) using the migrated SQL.
- ❑ Build the IMAGE emulator database (if applicable).
- ❑ Build the PowerHouse dictionary using the migrated PDL.
- ❑ Create the indexed, sequential, and direct files using QUTIL.
- ❑ Load the data into the file systems using QTP and the migrated subfiles.
- ❑ Convert migrated UDCs into UNIX scripts or Windows command files, as appropriate. For more information, see (p. 16) for UNIX or (p. 28) for Windows.
- ❑ Compile and test the pilot application, modifying the data definitions and programs as needed to fit the target platform and file system(s).

- ❑ Modify the data definitions and programs as desired to take advantage of platform or file system features that were not available on the original platform.
- ❑ Apply system, application, and data security (if part of your pilot application).

Step 8: Assess Steps 2 through 7

Assess your pilot application by asking yourself:

- Is the application definition (Step 1) appropriate?
- Is your inventory (Step 3) complete?
- Did your plan (Step 4) omit anything?
- Did things move as expected?

This is the stage where you modify your activities in Steps 2 through 7 in order to arrive at a pilot application that is ready for a user evaluation. It may require several cycles before proceeding to Step 9. At this point, you may also want to consider volume testing and/or performance testing.

Step 9: Have Users Evaluate the Pilot

Testing for user acceptance on a pilot application can identify issues that will save you much effort when it comes to migrating the full application. Among other things, ask:

- Are the expectations of the users being met?
- Are the expectations of the users realistic?
- Are your UI standards appropriate?
- Do the users need more training?
- Is the security appropriate?

As you test for user acceptance, you will probably uncover issues about user requirements and the user interface in the new environment. Early user involvement also ensures a higher level of user acceptance of the new application, as they have had direct involvement throughout the process.

Step 10: Assess Steps 2 through 9

Modify the activities in Steps 2 through 9 based on the feedback from your users, and also based on what you (the migration team) have learned during the pilot migration. For example, you may want to look at:

- additional training, hardware, or software
- changes in file systems, server topology, or user interface

Step 11: Plan the Complete Migration

Now that you know what is involved in such a migration, you can make decisions about the complete application. We recommend delaying these final decisions until this point in the process, since they can be affected by discoveries during the pilot migration.

Step 12: Do the Complete Migration

Incorporating any planning changes made during Step 11, you are now ready to apply the 12-step process to the migration of the complete application.

In Conclusion

The key decisions to make in planning your migration from MPE/iX are:

- **Target platform**: UNIX or Windows
- **Target file systems:** IMAGE emulators, relational, indexed, direct or sequential
- **End-user UI**: Terminal, Windows GUI (using Axiant), or Web Browser (using PowerHouse Web)
- **Migration method:** using Axiant migration tools or using editors and UNIX scripts
- **Migration timing:** what do you move and in what sequence?

You now have the resources to make these decisions.

Index