Cognos(R) Application Development Tools

PowerHouse(R) 4GL

VERSION 8.4E

PDL AND UTILITIES REFERENCE

COGNOS®

THE NEXT LEVEL OF PERFORMANCE™

Product Information

This document applies to PowerHouse(R) 4GL Version 8.4E 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 is intended for Cognos PowerHouse users who use PDL to create PowerHouse dictionaries. It also provides detailed information about dictionary-related utilities.

Chapter 1, "Introducing the PowerHouse Dictionary", introduces PDL and the other PowerHouse components and utilities.

Chapter 2, "PDL Statements", provides concise summaries and detailed information about PDL statements. Syntax summaries, detailed syntax discussions, and examples are provided for each PDL statement, where applicable.

Chapter 3, "Standard Elements and Usages", is a summary of the attributes of standard usages provided when you create a dictionary with the PRELOADED option.

Chapter 4, "QSHOW Statements", provides concise summaries and detailed information about QSHOW statements. Syntax summaries, detailed syntax discussions, and examples are provided for each QSHOW statement, where applicable.

Chapter 5, "QUTIL Statements", provides concise summaries and detailed information about QUTIL statements. Syntax summaries, detailed syntax discussions, and examples are provided for each QUTIL statement, where applicable.

Chapter 6, "ITOP Utility", explains how to use the IMAGE to PDL conversion utility to generate PDL statements directly from an existing IMAGE database.

Chapter 7, "ETOP Utility", explains how to use the Eloquence to PDL conversion utility to generate PDL statements directly from an existing Eloquence database.

Chapter 8, "QCOBLIB Utility", documents how to use the QCOBLIB utility to generate COBOL definitions from a PDL dictionary.

Chapter 9, "PH Integrator Utility", describes how to use the PowerHouse Integrator utility to assist in the translation of the CDD/Repository and PowerHouse dictionary.

Conventions in this Book

This book is for use with MPE/iX, OpenVMS, UNIX, and Windows operating systems. Any differences in procedures, commands, or examples are clearly labeled.

In this book, words shown in uppercase type are keywords (for example, SAVE). Words shown in lowercase type are general terms that describe what you should enter (for example, filespec). When you enter code, however, you may use uppercase, lowercase, or mixed case type.

The term PDL can refer to the PowerHouse component, the PowerHouse Definition Language, and the PDL and PHDPDL executables used to compile PowerHouse source statements. Whenever it is necessary to differentiate between the compilers, the terms PDL and PHDPDL are used.

Getting Help

For more information about using this product or for technical assistance, visit the Cognos Global Customer Services Web site (http://support.cognos.com). This site provides product information, services, user forums, and a knowledge base of documentation and multimedia materials. To create a case, contact a support person, or provide feedback, click the **Contact Us** link at the bottom of the page. To create a Web account, click the **Web Login & Contacts** link. For information about education and training, click the **Training** link.

Cognos PowerHouse 4GL Documentation Set

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

Objective	Document
Install PowerHouse 4GL	*Cognos PowerHouse 4GL & PowerHouse Web Getting Started* book. This document provides step-by-step instructions on installing and licensing PowerHouse 4GL. Available in the release package or from the following website: http://support.cognos.com
Review changes and new features	*Cognos PowerHouse 4GL & PowerHouse Web Release and Install Notes*. This document provides information on supported environments, changes, and new features for the current version. Available in the release package or from the following website: http://support.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 the following website: http://powerhouse.cognos.com

Objective	Document
Get detailed reference information for PowerHouse 4GL	Cognos PowerHouse 4GL Reference documents. They provide detailed information about PowerHouse rules and each PowerHouse component. The documents are • *Cognos PowerHouse 4GL PowerHouse Rules* • *Cognos PowerHouse 4GL PDL and Utilities Reference* • *Cognos PowerHouse 4GL PHD Reference* • *Cognos PowerHouse 4GL PowerHouse and Relational Databases* • *Cognos PowerHouse 4GL QDESIGN Reference* • *Cognos PowerHouse 4GL QUIZ Reference* • *Cognos PowerHouse 4GL QTP Reference* Available from the PowerHouse 4GL documentation CD or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com

Cognos PowerHouse Web Documentation Set

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

Objective	Document
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 PowerHouse Web Administrator CD or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com
Install PowerHouse Web	*Cognos PowerHouse 4GL & PowerHouse Web Getting Started* book. This document provides step-by-step instructions on installing and licensing PowerHouse Web. Available in the release package or from the following website: http://support.cognos.com

Objective	Document
Review changes and new features	*Cognos PowerHouse 4GL & PowerHouse Web Release and Install Notes*. This document provides information on supported environments, changes, and new features for the current version. Available in the release package or from the following website: http://support.cognos.com
Get detailed information for developing PowerHouse Web applications	*Cognos PowerHouse Web Developer's Guide*. This document provides detailed reference material for application developers. Available from the Administrator CD or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com
Administer PowerHouse Web	The *PowerHouse Web Administrator Online Help*. This online resource provides detailed reference material to help you during PowerHouse Web configuration. Available from within the PowerHouse Web Administrator.

Cognos Axiant 4GL Documentation Set

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

Objective	Document
Install Axiant 4GL	*Cognos Axiant 4GL Web Getting Started* book. This document provides step-by-step instructions on installing and licensing Axiant 4GL. Available in the release package or from the following website: http://support.cognos.com
Review changes and new features	*Cognos Axiant 4GL Release and Install Notes*. This document provides information on supported environments, changes, and new features for the current version. Available in the release package or from the following website: http://support.cognos.com

Objective	Document
Get an introduction to Axiant 4GL	*A Guided Tour of Axiant 4GL*. This document contains hands-on tutorials that introduce the Axiant 4GL migration process and screen customization. Available from the Axiant 4GL CD or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com
Get detailed reference information on Axiant 4GL	*Axiant 4GL Online Help*. This online resource is a detailed reference guide to Axiant 4GL. Available from within Axiant 4GL or from the following websites: http://support.cognos.com and http://powerhouse.cognos.com

For More Information

For information on the supported environments for your specific platform, as well as last-minute product information or corrections to the documentation, see the *Release and Install Notes*.

Chapter 1: Introducing the PowerHouse Dictionary

Overview

This chapter introduces PDL and its utilities. It also provides overview information about other PowerHouse components and utilities, and about how a dictionary stores information.

About PowerHouse

PowerHouse 4GL is an application development environment that allows you to create business applications quickly and easily.

Components

PowerHouse 4GL is divided into the following separate, yet integrated components:

PowerHouse Dictionary

The PowerHouse dictionary is the foundation of PowerHouse applications. As the backbone of all PowerHouse systems, the PowerHouse dictionary stores definitions of the data used by your PowerHouse applications.

There are two dictionary types—PDC and PHD. PDC dictionaries exist as a single file and have a .pdc extension (**OpenVMS, UNIX, Windows**) or file code 655 (**MPE/iX**). PHD dictionaries exist as five indexed files and have a .phd extension. PHD dictionaries are OpenVMS-specific.

For more information about the PHD dictionary, see the *PHD Reference* and *PowerHouse Rules books*, and the section, "PowerHouse Dictionary on OpenVMS" on (p. 18).

PDL

The PowerHouse Definition Language (PDL) allows you to create and maintain a PowerHouse dictionary.

PDL source code can be compiled in either the PDL or PHDPDL (**OpenVMS**) compiler.

PDL Compiler

PDL compiler is the component that compiles PDL source statements to a PowerHouse dictionary. Dictionaries generated with the PDL compiler have a .pdc extension (**OpenVMS, UNIX, Windows**) or file code 655 (**MPE/iX**).

PHDPDL Compiler (OpenVMS)

PHDPDL is an OpenVMS-specific component that compiles PDL source statements to a PowerHouse dictionary. Dictionaries generated with PHDPDL have a .phd extension.

For more detailed information on PHDPDL, see the section, "PowerHouse Dictionary on OpenVMS", on (p. 18).

PHD Screen System (OpenVMS)

PHD is a screen interface to PHD dictionaries. You can initiate PHD with the POWERHOUSE or POW command.

For more information about running PHD, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules book*.

QDESIGN and QUICK

QUICK is an interactive screen processor with a powerful development tool: QDESIGN. As a screen designer, you use QDESIGN to build data entry and retrieval screen systems. QUICK screens are used by data-entry operators and other end-users to process data quickly or to browse effortlessly through their files.

QUICK includes an interactive debugger that lets you analyze and control QUICK screens as they run.

QUIZ

QUIZ is the PowerHouse report writer. It takes the information you request and gives it a structure. Your information is automatically displayed in columns with headings. The key to the simplicity of QUIZ lies in its relationship with the data dictionary. QUIZ references the rules and standards defined in the data dictionary by the application designer when it formats your report.

QTP

QTP is a high-volume transaction processor. It gives you the power to change the data in your files in one sweep. Because QTP is easy to use and designed for fast, high-volume file updating, it should be used by someone who is familiar with the implications of updating active files.

QTP includes a trace facility that lets you debug QTP requests.

Utilities

PowerHouse also contains the following data dictionary utilities:

QSHOW

QSHOW is the data dictionary reporting program. It allows you to view and obtain cross-reference information about the contents of your PowerHouse dictionaries. It also allows you to generate PDL source for a PowerHouse dictionary.

QUTIL

QUTIL is a utility that creates and deletes non-relational files and databases.

ITOP (MPE/iX)

ITOP is a IMAGE to PDL conversion utility that generates PDL statements directly from an existing IMAGE database.

QCOBLIB (MPE/iX)

QCOBLIB is a utility that generates COBOL definitions from a PDL dictionary.

PHDMAINTENANCE (OpenVMS)

PHDMAINTENANCE creates and manages PHD dictionaries. It is also referred to as PHDMAINT.

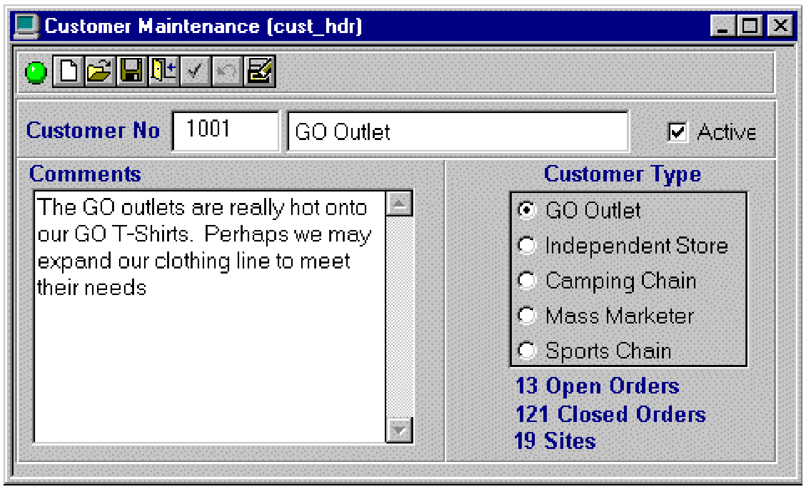
PHDADMIN (OpenVMS)

PHDADMIN is a run-time utility for administering security classes in PHD dictionaries.

PowerHouse-Related Products

Axiant 4GL

Axiant 4GL is a visual Windows-based development environment for creating PowerHouse applications. With Axiant 4GL, you can build applications that can be deployed in a variety of thin-client, fat-client, mobile, stand-alone, and server-only architectures. Axiant 4GL gives PowerHouse a Windows-like user interface.



Customer Maintenance (cust_hdr)

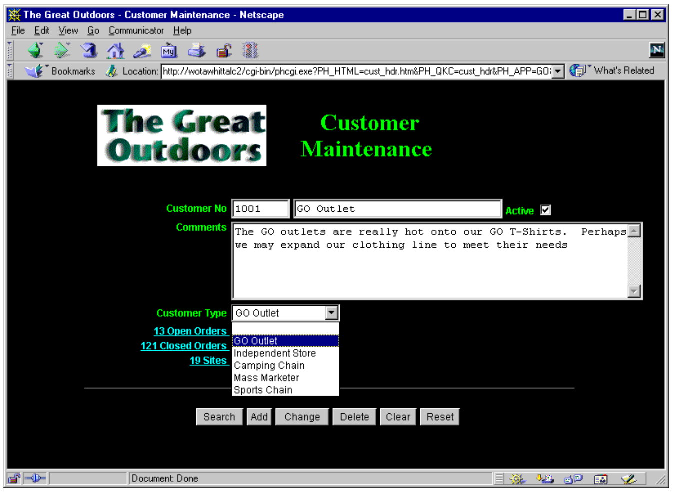
Customer No: 1001 | GO Outlet | Active ✓

Comments: The GO outlets are really hot onto our GO T-Shirts. Perhaps we may expand our clothing line to meet their needs

Customer Type: [✓ GO Outlet] Independent Store / Camping Chain / Mass Marketer / Sports Chain

13 Open Orders
121 Closed Orders
19 Sites

PowerHouse Web



The Great Outdoors - Customer Maintenance - Netscape

Location: http://wotawhittalc2/cgi-bin/phcgi.exe?PH_HTML=cust_hdr.htm&PH_QKC=cust_hdr&PH_APP=GO:

Customer Maintenance

Customer No: 1001 | GO Outlet | Active ✓

Comments: The GO outlets are really hot onto our GO T-Shirts. Perhaps we may expand our clothing line to meet their needs

Customer Type: GO Outlet (GO Outlet, Independent Store, Camping Chain, Mass Marketer, Sports Chain)

13 Open Orders
121 Closed Orders
19 Sites

Document: Done

PowerHouse Dictionary and Entities

Dictionary

A PowerHouse dictionary contains information about the entities that PowerHouse applications manipulate. PowerHouse components and utilities use the information in the dictionary to access and manipulate data.

Entities

The PowerHouse entities are as follows:

Entity	Description
Usage	A usage is a template for defining elements. A single usage can serve as a template for many element definitions.
Element	An element describes the attributes of a class of data. Each element must have a type (character, numeric, or date), a size, and a unique name. An element also has a display format and other attributes.
File	A file is a set of records. Each file in a dictionary has a unique name, which is not necessarily the same as the operating system's name for the file. Each file has one or more record-structures associated with it.
Record	A record names a record-structure. A record-structure is an ordered collection of items. Each record-structure has a unique name. Each record-structure is associated with exactly one file.
Item	An item is an entity in a record-structure that holds a value. Each item has the same name as an element, and has that element's attributes.
Index	An index helps locate data quickly and provides a way to link records in one file with those in another. Indexes can be either primary or alternate. When an indexed file is defined, one index is the primary index and the others are the alternate indexes. By default, the first index is the primary index but you can override the default and specify a different primary index. Indexes can be unique or repeating. In a unique index, every record will have a unique set of values for the items that are segments. In a repeating index, two or more records may have the same set of values for the items that are segments.
Segment	In PowerHouse, an index is composed of one or more segments. A segment is an item that is part of an index. The order of segments is significant. An INDEX statement must be followed immediately by at least one SEGMENT statement. The SEGMENT statements define the items that are included in the index.
Database	A database names and gives attributes to a relational database.
Transaction	A transaction is a group of operations treated as a unit by a relational database. All database activities take place within one or more transactions.

PDL

PDL, PowerHouse Definition Language, is an English-like language that allows you to enter definitions for your data in straightforward statements. PDL statements are parsed by the PDL compilers.

With the PDL or PHDPDL compiler, you can

- create a dictionary
- define entities for a new dictionary
- add entities to an existing dictionary
- compile a dictionary

The Order of PDL Statements

PDL statements must be written in a specific order. The basic statements, in order, for defining a dictionary are

CREATE DICTIONARY

USAGE

ELEMENT

FILE

RECORD

ITEM

INDEX

SEGMENT

DATABASE

TRANSACTION

LOAD

All statements between the CREATE DICTIONARY statement and the LOAD statement belong to one dictionary.

USAGE statements must be defined before ELEMENT statements so that they can be used in ELEMENT statements to set element attributes to predetermined values.

A dictionary contains one or more ELEMENT statements. ELEMENT statements must be entered before any record-structure that references them.

A FILE statement can come after any ELEMENT statement.

Each FILE statement is followed by one or more RECORD statements. The RECORD statements that follow a FILE statement define record-structures that belong to that FILE.

Each RECORD statement is followed by one or more ITEM statements. Every item must have the same name as an ELEMENT that has already been defined. The order of the ITEM statements defines the order in which the data will be stored in the records in the file. (BEGIN STRUCTURE and END STRUCTURE statements may appear among the ITEM statements.)

The ITEM statements are followed by zero or more INDEX statements. The index applies to the record-structure, and through the record-structure to the file.

Each INDEX statement is followed by one or more SEGMENT statements. The SEGMENT statements define the items that are part of the index.

The DATABASE statement can be entered at any time.

The TRANSACTION statement must follow any DATABASE statement that it refers to. Each TRANSACTION statement defines a transaction and its attributes.

You can use the REVISE, SAVE, SET, and USE statements anywhere in a PDL session without affecting the order given above.

Using Existing Dictionaries

PDL and PHDPDL allow you to add new elements, files, and record-structures to an existing PDC or PHD dictionary, respectively. PDL and PHDPDL do not allow you to modify or delete existing entities in the dictionary. With PDL, you must be the dictionary creator to add new dictionary entities to an existing dictionary. With PHDPDL, you must be the dictionary creator, or if dictionary security is specified, you must be a dictionary manager to add new dictionary entities.

To access an existing dictionary, use the SET DICTIONARY statement in place of the CREATE DICTIONARY statement. Then define the new entities as you would if you were defining a new dictionary.

You can use existing element definitions in new ITEM statements. You can use existing usage definitions in new ELEMENT statements.

You can add new record-structures to an existing file by giving a FILE statement that refers to an existing file, followed by the RECORD statement. You must give the file's name and organization exactly as they appeared in the original definition.

PowerHouse Dictionary on OpenVMS

PowerHouse on OpenVMS includes additional components that let you create and manage a PHD dictionary. These components are the PHD Screen System, the PHDPDL compiler, PHDMAINT, and PHDADMIN. With these components, you can

- use PHDMAINT or PHDPDL to create a PHD dictionary
- use the PHDPDL compiler to compile the PHD dictionary using PDL syntax
- update the PHD dictionary interactively using the PHD Screen System and PHDADMIN

Dictionary Types

There are two dictionary types available on OpenVMS. They are the PHD and PDC dictionaries. When choosing the dictionary type, it is important to consider the differences between the PHD and PDC dictionaries. Other considerations to take into account when choosing the dictionary type are the size of the dictionary and the amount of the dictionary that is actually used by each user or application.

The PHD and PDC dictionary differ in

- the physical structure of the file
- dictionary access
- dictionary maintenance

The PHD dictionary is comprised of five indexed files—the traditional PowerHouse dictionary format on OpenVMS. These indexed files contain dictionary information that is updateable and accessible using indexed retrieval. Indexed retrieval lets you obtain specific pieces of information when needed.

The PDC dictionary exists in the platform-generic format of a single file. It is not updateable and is accessed by being loaded into memory in its entirety when PowerHouse is started. The PDC dictionary can make use of Global Sections to optimize the memory load that a large dictionary of this type can put on the system. Using Global Sections allows the dictionary memory to be shared among users. Because the dictionary is all in memory, initial access may be slightly slower while general access may be faster than that provided by indexed access.

PH_CREATE_SHARED Logical

The PDC type dictionary is read into memory when the product is activated. For large dictionaries and large numbers of users, this can cause excessive memory usage.

To address this issue, you can use the logical PH_CREATE_SHARED to cause the dictionary to be loaded into shared memory using global sections in the same manner as QUICK screens can be shared. This logical is defined in PH_LOCATION:SETPH.COM to "G" or Group sharing, the default. The logical can be changed locally.

The valid options are "P" for Private, "G" for Group and "S" for System. "S" requires privilege to implement. In QUICK, the QKGO setting for screen sections also affects the sharing of the dictionary. For more information, see the *Cognos PowerHouse 4GL Getting Started* book.

For more information about the PDC and PHD dictionaries, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules* book.

Dictionary Maintenance

There are two ways to maintain dictionaries on OpenVMS. One method is to use the syntax-based compilers, PDL and PHDPDL. PDL is used with PDC dictionaries and PHDPDL is used with PHD dictionaries. The compilers let you add new information to an existing dictionary but do not let you modify information that already exists in the dictionary. To modify information using the syntax-based compilers, the entire dictionary must be recreated.

The other method of maintaining a PHD dictionary is to use the menu-driven QUICK application, the PHD Screen System. The PHD Screen System allows you to maintain a PHD dictionary without knowing any syntax. The information you add is inserted directly into the dictionary and requires no recompilation. Information in the dictionary can be modified as desired. The PHD Screen System does not support PDC dictionaries.

Whether you use the syntax-based compilers or the PHD Screen System to maintain your dictionary, it is important to remember that screens, reports, and runs must be recompiled in order to pick up changes made in the dictionary.

PDL syntax has some restrictions based on the compiler being used. PHDPDL accepts all syntax with the exception of the SET NO|UP|DOWNSHIFT statement which it accepts, but ignores. PDL accepts, but ignores, the syntax relating to dictionary security, as this syntax applies only to PHD dictionaries.

The PHD Screen System allows you to add incomplete definitions to the dictionary, whereas the syntax-based compilers do not. Incomplete definitions are allowed even though they can create inconsistencies in the dictionary should you try to use them. For example, with the PHD Screen System, it is possible to create an indexed file definition without specifying an index. You can't create or use the indexed file without the index, but PHD doesn't stop you from creating the incomplete definition until you need to use it. PDL and PHDPDL do not allow these inconsistencies. You get errors in PDL or PHDPDL if you try to do this.

The PHD Screen System and the syntax-based compilers handle ELEMENT types and sizes differently. Cases exist where the PHD Screen System allows for softer rules regarding ELEMENT types and sizes. PHDPDL tolerates these issues and does not produce warnings whereas PDL (which was not designed to co-exist with PHD) produces warnings.

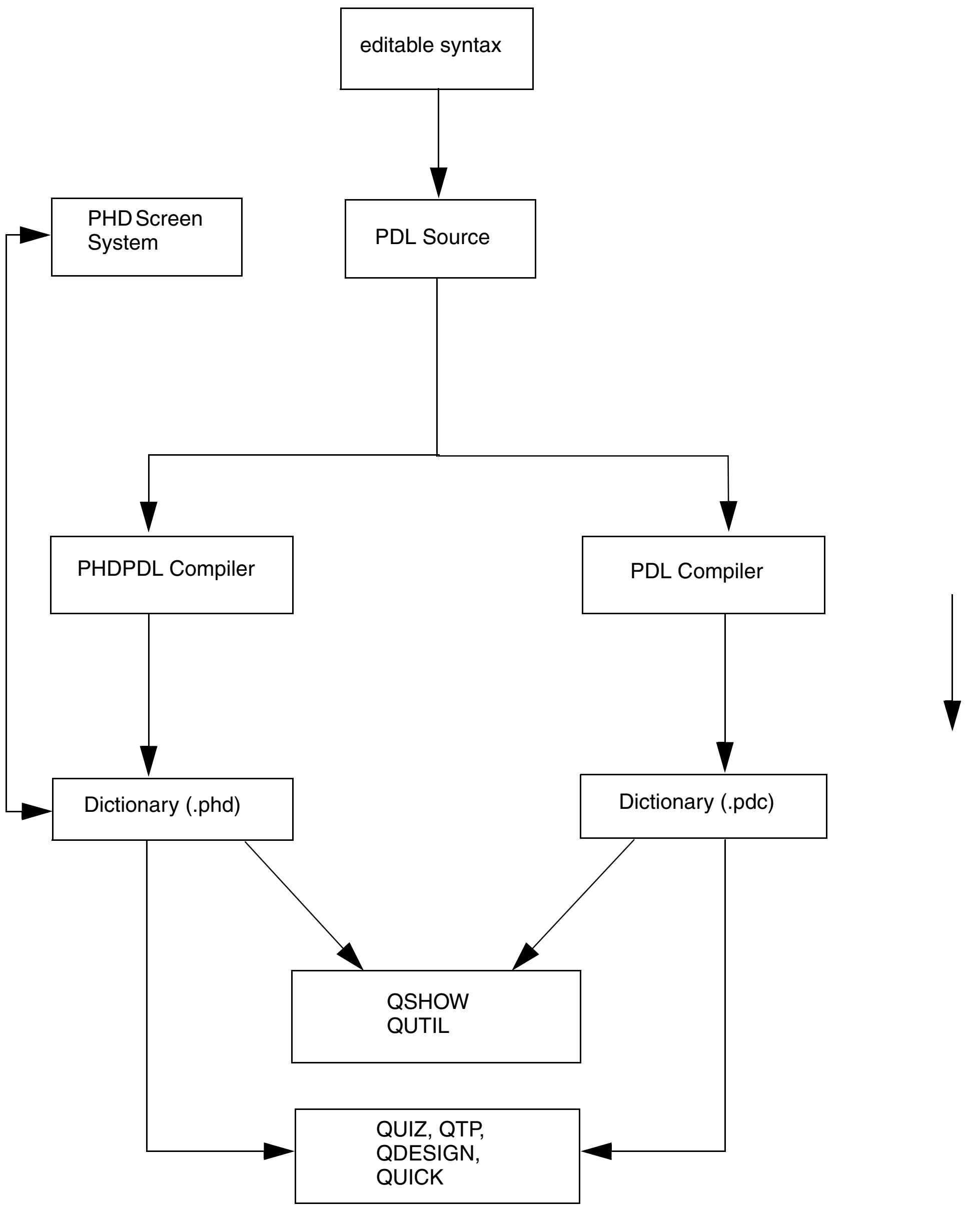
Differences Between PDL, PHDPDL, and the PHD Screen System

The following table describes the main differences between the syntax-based compilers PDL and PHDPDL and the menu-driven interface, the PHD Screen System:

PDL/PHDPDL	**PHD Screen System**
designed to be used primarily in batch mode	designed to be used interactively
provides a command-driven approach to data definition, requiring you to define each entity explicitly and in a particular order	provides a form-oriented approach to data definition and helps you along by prompting you and by defining certain entities automatically
provides a quick, simple interface for the syntax-proficient	provides a productive interface for the novice user
can't create a data file. Use QUTIL.	uses the File, Records, or File Maintenance Screen to create data files
can take definitions in the form of PDL statements from one dictionary and then load them into another dictionary all at once	you must enter each definition separately on a screen
lets you enter new data definitions into a dictionary. PDL doesn't let you change or delete most existing definitions without recompiling the entire dictionary. You can add new entities without recompiling.	lets you enter new data definitions into a dictionary. PHD lets you easily change options.
provides syntax help	has extensive online help messages and explanations
provides a way to back up a PowerHouse dictionary with verifiable source statements	source statements can be generated using QSHOW for use as a backup

PowerHouse Components on OpenVMS

The following diagram shows the relationship between the various PowerHouse components on OpenVMS.



Chapter 2: PDL Statements

Overview

This chapter describes each of the PDL statements and options in detail. Syntax descriptions and statement discussions are provided for each statement. Where applicable, examples are also provided.

PDL uses the syntax and language rules discussed in Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Summary of PDL Statements

Statement	Purpose
APPLICATION SECURITY CLASS	Defines an application security class (ASC) and assigns members to the ASC.
BEGIN STRUCTURE	Begins an item substructure.
CANCEL	Cancels the current set of dictionary design specifications.
CREATE DICTIONARY	Creates a new dictionary or replaces an existing dictionary.
DATABASE	Identifies a relational database.
DICTIONARY SECURITY CLASS (**OpenVMS**)	Names a dictionary security class (DSC) and assigns members to the DSC.
DESCRIPTION	Provides a description of an application security class, dictionary security class (PHDPDL) element, file, record, or usage.
ELEMENT	Declares an element.
END STRUCTURE	Ends an item substructure.
EXECUTION-TIME PARAMETERS (**OpenVMS**)	Defines a default location for data files.
EXIT	Ends a PDL and PHDPDL session.
FILE	Defines a file and its attributes.
INDEX	Declares an index in a record-structure.
ITEM	Declares an item.
LOAD	Loads the current PDL definitions into the dictionary.
PERMIT	Gives or denies an application security class access to a record-structure or element.

Statement	**Purpose**
QUIT	Ends a PDL and PHDPDL session.
RECORD	Declares a record-structure.
REVISE	Invokes the system editor to edit PDL source statement files.
SAVE	Saves the current PDL statements in a file.
SEGMENT	Declares a segment in an index.
SET	Changes the default settings for a session.
SHOW DICTIONARY	Displays the name of the current dictionary.
SYSTEM OPTIONS	Determines system-wide options and defaults for PowerHouse applications.
TRANSACTION	Defines transactions used for relational files.
USAGE	Declares a usage.
USE	Processes PDL statements contained in source statement files.
USER MODE (**OpenVMS**)	Defines a user mode and assigns its members.

APPLICATION SECURITY CLASS

Defines an application security class (ASC) and assigns members to the ASC.

Syntax

APPLICATION SECURITY CLASS name [option]...

ASC name [option]...

name

Names a new ASC, or one that has already been defined.

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters defined in the dictionary by the SPECIAL NAME CHARACTERS option of the SYSTEM OPTIONS statement.

Options

Application Security Class (ASC) Options		
DESCRIPTION	LOGONID	PASSWORD
PORTID	UIC	WEBLOGONID

DESCRIPTION string [[,] string]...

The strings contain a description of the ASC. The description can be reported by QSHOW reports.

Limit: 60 characters per string. Each string must be enclosed in quotation marks.

LOGONID logonid [[,]LOGONID] logonid]...

Assigns user names that are potential members of the application security class LOGONID.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is LOGONID, and if the user's logon ID is listed in a LOGONID option of an ASC statement, the user becomes a member of the ASC. Otherwise, the user becomes a member of ASC UNKNOWN.

MPE/iX: Logonid is defined in the form username.account.

OpenVMS: A logonid is an OpenVMS user name with a limit of 12 characters.

UNIX: A logonid is a UNIX logonid.

Windows: A logonid is a Windows user name.

For more information, see the PERMIT statement (p. 79).

Limit: Logonids must be enclosed in quotation marks if they begin with a numeric.

PASSWORD password [[,] password]... (OpenVMS)

Assigns passwords that the user may enter to become a member of the application security class PASSWORD.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is PASSWORD, PowerHouse prompts the user for a password. If the user enters one of the passwords listed after the PASSWORD option of an ASC statement, the user becomes a member of that ASC. Otherwise, the user becomes a member of the ASC UNKNOWN.

Limit: 64 characters. Enclose the password in quotation marks if it contains characters other than letters, digits, or the special name characters defined in the dictionary by the SPECIAL NAME CHARACTERS option of the SYSTEM OPTIONS statement.

PORTID portid [[,] portid]... (OpenVMS)

Assigns port IDs to an ASC. A portid is an OpenVMS device name. A user who accesses PowerHouse through one of the port IDs listed may become a member of this ASC.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is PORTID, and if the user's port ID is listed in an ASC statement, the user becomes a member of that ASC. Otherwise, the user becomes a member of the ASC UNKNOWN.

Limit: 16 characters. The port IDs can be listed either with or without quotation marks around each one.

UIC [gid,uid] [[,] [gid,uid]]... (UNIX)
UIC [gid,mid] [[,] [gid,mid]]... (OpenVMS)

Lists members of the ASC by their UICs:

OpenVMS:	The UIC is defined by a gid and a mid (member id number). The gid can be from 0-37776 (octal) and the mid can be from 0-177776 (octal). The gid and mid must be declared as a pair.
UNIX:	The UIC is defined by a gid (group id number) and a uid (user id number). The gid and uid can be any number from 1 to 30,000 and must be declared as a pair.

The brackets around the [gid,uid] and [gid,mid] portion are required syntax.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is UIC, and if the user's UIC is listed in an ASC statement, the user becomes a member of that ASC. Otherwise, the user becomes a member of the ASC UNKNOWN.

WEBLOGONID username [[,] username]...

Assigns user names that are potential members of this ASC. Username is case sensitive.

When PowerHouse Web opens a dictionary it checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is WEBLOGONID, PowerHouse Web checks authenticated usernames against the list of usernames in the WEBLOGONID option. If the authenticated username is listed in a WEBLOGONID option of an ASC statement, the user becomes a member of that ASC and the KNOWN class. If an authenticated username is not listed in any WEBLOGONID option of any ASC statement, the user is treated as a member of the ANONYMOUS class and the UNKNOWN class.

If there is no authenticated username, or the ASC ID METHOD is not WEBLOGONID, the user becomes a member of the ASC UNKNOWN.

Discussion

An application security class is a group of PowerHouse users who share the same access (through PowerHouse) to the data used in your application. QUICK, QUIZ, and QTP automatically enforce the data security provisions specified in the data dictionary, based on your application security class. Application security does not override operating system security.

All ASC options may be placed in the dictionary. However, the active security option depends on the ASC ID METHOD option specified in the SYSTEMS OPTIONS statement.

If a user is associated with a given application security class that allows access to a specific file, the user can access that file. If the user is not associated with an application security class that has access to a specific file, access is denied. Access to individual files and/or record items can be restricted to a subset of users.

You can assign the same user to more than one APPLICATION SECURITY CLASS statement. If the same user is assigned to more than one ASC with the same ID method, the user becomes a member of all those ASCs. The user's access rights are the union of the access rights of each ASC.

You can't assign the same user more than once to the same ID method of the same ASC statement.

The following information is compiled into QUIZ reports, QTP runs, and QUICK screens:

- ASC ID Method
- the name of the ASC and the association of the ASC with records and elements
- User Modes and location specified in the ETP statement (**OpenVMS**)

If any of this information changes, the corresponding screens, reports, and runs need to be recompiled.

OpenVMS: For PHD dictionaries, the membership of ASCs and User Modes is dynamic. You can add to and delete IDs from ASCs without requiring that screens, reports, and runs be recompiled.

UNKNOWN Security Class

A default class can be used for dealing with PowerHouse users who aren't included in any of the security classes of the USER statements. In most cases, any PowerHouse user not included in a security class list is considered a member of the application security class, UNKNOWN. This is a special application security class that you can specify in a PERMIT statement. It always exists in PowerHouse.

Members of the UNKNOWN application security class are given access to data for which security isn't specified, or for which the security class is unknown. If unidentified users are using PowerHouse applications, it's useful to specify security for the UNKNOWN class. The simplest way to declare security for the UNKNOWN class is to specify the class, UNKNOWN, in the list of classes in each PERMIT statement.

SAFETY-NET Security Class

A simple way to ensure the security of your sensitive data is to use the preloaded class called SAFETY-NET. The SAFETY-NET class has no members. Specify that SAFETY-NET has read access to all sensitive record-structures and elements. This way, all other ASCs which are not explicitly given access to the sensitive data will be denied access to it.

Examples

In the following examples, no other PERMIT statements reference the INVENTORY record structure or elements in the INVENTORY record. In these example:

- PowerHouse components use application security classes combined with PERMIT statements to determine access rights to record-structures and elements.
- The users with LOGON IDs, "PRG" and "MANAGER", can read or write the INVENTORY record-structure.
- User "CLERK" can read the INVENTORY record-structure, but can only write to the QTYONHAND field of the INVENTORY record-structure
- No other users can access the INVENTORY record-structure.

MPE/iX

```
> SYSTEM OPTIONS &
>      ASC ID METHOD LOGONID
.
.
.
> ASC DEVELOPMENT &
>      LOGONID PRG.DOC
>
> ASC CLERK &
>      LOGONID CLERK.DOC
>
> ASC PRODUCTION &
>      LOGONID MANAGER.DOC &
>            LOGONID PRG.DOC
.
```

```
.
.
> PERMIT WRITE &
>      OF RECORD INVENTORY BY &
>      DEVELOPMENT &
>      PRODUCTION
>
> PERMIT READ &
>      OF RECORD INVENTORY BY &
>      CLERK
>
> PERMIT CLERK TO &
>      READ &
>      ELEMENT &
>         INVENTORY-NUM &
>         INVENTORY-ITEM &
>         UNIT-COST &
>         UNIT-MARKUP
>
> PERMIT CLERK TO &
>      CHANGE &
>      ELEMENT &
>         QTY-ON-HAND
```

UNIX

```
> SYSTEM OPTIONS &
>      ASC ID METHOD UIC
.
.
.
> ASC DEVELOPMENT &
>      UIC [26,123] [26,128]
>
> ASC CLERK &
>      UIC [26,185]
>
> ASC PRODUCTION &
>      UIC [26,190] &
>            [26,195]
.
.
.
> PERMIT WRITE &
>      OF RECORD INVENTORY BY &
>      DEVELOPMENT &
>      PRODUCTION
>
> PERMIT READ &
>      OF RECORD INVENTORY BY &
>      CLERK
>
> PERMIT CLERK TO &
>      READ &
>      ELEMENT &
>         INVENTORYNUM &
>         INVENTORYITEM &
>         UNITCOST &
>         UNITMARKUP
>
> PERMIT CLERK TO &
>      CHANGE &
>      ELEMENT &
>         QTYONHAND
```

OpenVMS

```
> SYSTEM OPTIONS &
>      ASC ID METHOD LOGONID
```

```
.
.
.
> ASC DEVELOPMENT &
> PASSWORD "dev"
>   LOGONID PROGRAMMER

> ASC CLERK &
> PASSWORD "entry" &
>   LOGONID CLERK &
>      UIC [26,12] [26,12]
>
> ASC PRODUCTION &
> PASSWORD "prod" &
>        LOGONID MANAGER &
>        LOGONID PROGRAMMER &
.
.
> PERMIT WRITE &
>      OF RECORD INVENTORY BY &
>      DEVELOPMENT &
>      PRODUCTION
>
> PERMIT READ &
>      OF RECORD INVENTORY BY &
>      CLERK
>
> PERMIT CLERK TO &
>      READ &
>      ELEMENT &
>         INVENTORY_NUM &
>         INVENTORY_ITEM &
>         UNIT_COST &
>         UNIT_MARKUP
>
> PERMIT CLERK TO &
>      CHANGE &
>      ELEMENT &
>         QTY_ON_HAND
```

If, instead of the SYSTEM OPTIONS statement in the previous example, the dictionary contains

```
> SYSTEM OPTIONS &
>      ASC ID METHOD PASSWORD
```

the component prompts you for a password. If you enter "dev" or "prod" you can read or write the INVENTORY record-structure. If you enter "entry" you can read the INVENTORY record-structure, but can only change the QTY_ON_HAND field of the INVENTORY record-structure. If you enter anything else, you can’t access the INVENTORY record-structure at all.

BEGIN STRUCTURE

Begins an item substructure.

Syntax

BEGIN STRUCTURE

Discussion

Substructures allow you to deal with parts of an item instead of the entire item, without storing data twice.

The BEGIN STRUCTURE statement marks the beginning of a set of ITEM statements which are related as subitems to the last referenced ITEM statement. These relationships are generated until an END STRUCTURE statement is encountered.

Items in substructures can be subdivided into further levels of substructures. Each level of substructure is defined the same way using the BEGIN STRUCTURE and END STRUCTURE statements.

Each item declared in a substructure must correspond either to an ELEMENT statement that's already been declared or to an element that already exists in the dictionary. PowerHouse allows up to fifteen levels of substructures.

When you put a substructure on a numeric item, the item datatype must be ZONED. If you attempt to substructure a PACKED, INTEGER, or FREEFORM item, the substructure will not contain the correct portion of the data item.

The BEGIN STRUCTURE statement marks the beginning of a set of ITEM statements that define subitems in a substructure. The ITEM statements form a substructure of the item that was most recently defined before the BEGIN STRUCTURE statement. The END STRUCTURE statement must be used to mark the end of a block of ITEM statements in a substructure.

Example

This example shows how to access the year, month, and day of a date:

```
> ELEMENT DATEINVOICE DATE &
>      SIZE 8 FORMAT YYYYMMDD &
>      DEFAULT ITEM DATATYPE ZONED &
>      SIZE 8
.
.
.
> FILE INVOICEMASTER &
>      ORGANIZATION INDEXED &
>      OPEN INVOMAS
> RECORD INVOICEMASTER
>      ITEM INVOICENUM
>      ITEM CUSTOMER
>      ITEM EMPLOYEE
>      ITEM CUSTOMERORDNUM
>      ITEM TERMS
>      ITEM INVOICEPAID
>      ITEM DATEINVOICE DATATYPE ZONED SIZE 8
>         BEGIN STRUCTURE
>         ITEM DATEYEAR DATATYPE ZONED SIZE 4
>         ITEM DATEMONTHDAY DATATYPE ZONED SIZE 4
>            BEGIN STRUCTURE
>            ITEM DATEMONTH DATATYPE ZONED SIZE 2
>            ITEM DATEDAY DATATYPE ZONED SIZE 2
>            END STRUCTURE
>         END STRUCTURE
.
.
.
```

CANCEL

Cancels the current set of dictionary design specifications.

Syntax

CANCEL [CLEAR]

CLEAR

Clears the temporary source statement save file.

Discussion

The CANCEL statement cancels all previous statements in the current PDL or PHDPDL session, with the exception of SET statements.

All PDL statements that you enter are automatically stored in a temporary save file.

The CANCEL statement doesn't clear the source statements in the temporary save file unless the CLEAR option is used. PDL performs an implicit CANCEL after a LOAD statement.

By using the CLEAR option, you ensure that statements with errors aren't retained in the temporary save file. This is important when you save PDL source statements to permanent files.

The SAVE statement saves the contents of the temporary save file to a permanent file, including any erroneous statements. Your permanent source file will contain invalid PDL statements if the temporary save file contains invalid PDL statements and hasn't been cleared by the CLEAR option of the CANCEL statement.

The CANCEL CLEAR statement removes all of the statements that you've entered from the temporary save file. As a result, you must always re-enter all cleared statements after using the CANCEL CLEAR statement. To correct an improperly entered statement without clearing the temporary save file, use the REVISE statement to edit the contents of the temporary save file.

Example

The following example shows how CANCEL is used to correct errors in PDL design statements. It demonstrates how you can use the CANCEL CLEAR statement to clear the temporary save file when you've made an entry that PDL doesn't understand. In this example,

- All statements entered before CANCEL CLEAR are canceled.
- All statements after the CANCEL CLEAR statement are valid until the next CANCEL CLEAR statement is entered.

```
>  ELEMENT CUSTNAME CHARACTER SIZE 20
>  ELEMENT
.
.
.
>
>  FILE INVOICE ORGANIZATION INDEXED
>     RECORD INVOICE
>         ITTEM CUSTNAME
>         ^^^^^
*E* Expected: CANCEL CREATE EXIT QUIT REVISE SAVE
SET SHOW USE <eol> ITEM

> CANCEL CLEAR
>  ELEMENT CUSTNAME CHARACTER SIZE 20
.
.
.
>
>  FILE INVOICE ORGANIZATION INDEXED
>     RECORD INVOICE
>         ITEM CUSTNAME
```

```
>
.
.
.
> SAVE INVOICE
```

CREATE DICTIONARY

Creates a new dictionary or replaces an existing dictionary.

Syntax

CREATE DICTIONARY filespec [[NOT] PRELOADED]

filespec

Names the file in which the dictionary is stored.

NOT PRELOADED|PRELOADED

NOT PRELOADED indicates that the dictionary won't be preloaded with the standard dictionary usages and elements and security options.

PRELOADED indicates that the dictionary will be preloaded with the standard dictionary usages and elements and security options.

For information about creating your own usages, see (p. 118). For information about standard dictionary usages, see Chapter 3, "Standard Elements and Usages".

Default: PRELOADED

Discussion

The CREATE DICTIONARY statement creates a new dictionary. If the dictionary name you enter already exists, you receive the following prompt:

MPE/iX, UNIX, Windows:	`The file (name) already exists. Is it OK to delete? [Y/N]`
OpenVMS:	`The file already exists. Create new one? [Y/N]`

Note that when the dictionary is created, the dictionary name is used as the default dictionary title. You can set your own dictionary title using the SYSTEM OPTIONS TITLE statement. Although the dictionary is created immediately, dictionary definitions are loaded into the dictionary when you enter the LOAD statement.

The CREATE DICTIONARY statement issues an implicit CANCEL statement; any statements you enter prior to the CREATE DICTIONARY statement are therefore canceled.

OpenVMS: The dictionary logical(s) are set by this command to this dictionary.

Examples

In the following example, the CREATE DICTIONARY statement creates a dictionary called INVENTOR in the current directory:

```
> CREATE DICTIONARY INVENTOR
>
> ELEMENT ACCOUNTNUMBER NUMERIC SIZE 4
> ELEMENT LASTNAME CHARACTER SIZE 20
> ELEMENT FIRSTNAME CHARACTER SIZE 10
>
> FILE CUSTOMER ORGANIZATION INDEXED
> RECORD CUSTOMER
>         ITEM ACCOUNTNUMBER
>         ITEM LASTNAME
>         ITEM FIRSTNAME
>
>      INDEX ACCOUNTS UNIQUE
>         SEGMENT ACCOUNTNUMBER
>
> LOAD
```

Statements following CREATE DICTIONARY are saved in a temporary save file until the LOAD statement is entered. LOAD enters the definitions into the data dictionary named INVENTOR.

DATABASE

Identifies a relational database.

Syntax

DATABASE name TYPE type [option]...

name

A unique name given to the database to identify it to the PowerHouse dictionary. This name does not have to be the same as the database's physical name. For more information, see the OPEN option on (p. 34).

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters that are defined in the dictionary.

TYPE type

Type of relational database. The following table shows the supported relational database types for each platform.

MPE/iX:	ALLBASE
OpenVMS:	ORACLE, RDB, RDB/VMS
UNIX:	DB2, ODBC, ORACLE, SYBASE
Windows:	DB2, ODBC, ORACLE, SYBASE

ODBC

Windows: For Microsoft SQL Server, use ODBC. **UNIX:** PowerHouse supports the DataDirect ODBC connection to Microsoft SQL Server.

SYBASE

Accesses the database using the client library gateway (CTLIB).

RDB

Accesses the database through an SQL interface, using the PowerHouse data access layer. This allows you to use SQL statements and SQL-related syntax, such as the DECLARE CURSOR and CURSOR statements and the SQL-related verbs.

RDB/VMS

Accesses the database directly through the native API. There will likely be performance differences between this interface and the one used with TYPE RDB. The situation determines which interface is fastest.

Options

DATABASE Options		
`DEFAULT DATE [IS] NULL`	`DESCRIPTION`	`HIGH`
`LOW`	`NULL VALUES`	`OPEN`
`OWNER`	`PASSWORD`	`USERID`

DEFAULT DATE [IS] NULL

Enables NULL support for date fields. DEFAULT DATE [IS] NULL specifies that when a user enters a value of 0 in a date field, the date column is set to null. Setting the value to null lets users enter dates with a value of 0, but prevents PowerHouse from storing the date as 0 which is an invalid value in the database. Dates that are read as null are automatically converted to 0.

DESCRIPTION string [[,]string]...

Specifies a description that serves as documentation about a database. The description can be seen in QSHOW reports.

Limit: 60 characters per string.

HIGH [FILL] [HEXADECIMAL] string
LOW [FILL] [HEXADECIMAL] string

Specifies the high and/or low fill character used to formulate SQL queries for generic (@ and @@) search criteria.

If you use HEXADECIMAL, the string must be two hexadecimal digits. If you don't use HEXADECIMAL, the string must be a single ASCII character.

The defaults are database specific:

Database	HIGH fill character	LOW fill character
ALLBASE/SQL	HEX "FF"	HEX "00"
DB2	HEX "5A"	HEX "00"
MS SQL Server	HEX "7A"	HEX "00"
ODBC	HEX "FF"	HEX "00"
Oracle Rdb	HEX "FF"	HEX "00"
ORACLE	HEX "7F"	HEX "01"
SYBASE	the highest printable character on the machine	HEX "20"

For information about using the low and high fill options, see (p. 37).

NULL VALUES [NOT] ALLOWED

Determines whether PowerHouse enables null value support for a particular database. If support is enabled, PowerHouse maintains null values in expressions, conditions, and aggregate functions, and nulls are stored when items are not initialized. If support is disabled, null character items are initialized to spaces (or dictionary initial values), and null date or numeric items are initialized to zeros (or dictionary initial values).

Default: NULL VALUES NOT ALLOWED

OPEN filespec|open-name-string

Specifies the physical database that is accessed and associated with the name that identifies it to the PowerHouse dictionary.

Default: The PDL database name is used.

filespec

Specifies a valid file specification. It can be the physical name of the database as it is known to the operating system. For ALLBASE/SQL, the filespec must point to the root DBEnvironment.

Limits: For ALLBASE/SQL (**MPE/iX**) or Oracle Rdb (**OpenVMS**) databases only.

Default: The default filespec is the name after the DATABASE keyword.

open-name-string (OpenVMS, UNIX, Windows)

A string which is passed directly to the database server in order to gain access to the database.

An open-name-string contains delimited parameters such as userid, password, physical database name, network connection parameters, and possibly other parameters. If you include a userid and password in the open-name-string, it must resemble the following format:

SYBASE	<database physical name>@userid/password
ORACLE	ORACLE@userid[@<network connection parameters>]/password
DB2	not allowed
ODBC	not allowed

Please refer to your database documentation for more details on acceptable parameters, format, and syntax for your particular database's valid open specifications.

OpenVMS:	If the **close_detach** or **noprefix_openname** program parameter is used, the open-name-string can be a logical name containing the full open-name information.
UNIX, Windows:	The open-name-string may be an environment variable, which must be preceded by a dollar sign ($). If the string contains a required dollar sign, which is not used to specify an environment variable, use the backslash (\) to interpret it literally. For example: `ORACLE@OPS\$<userid>`

For ORACLE databases, the string, "ORACLE@", is inserted at the beginning of the supplied open-name-string if it does not exist.

Default: If no OPEN options are specified, the database server will look for default environment variables or logicals that are specific to running that database's environment.

Limit: For DB2, ODBC, ORACLE, and SYBASE databases only.

OWNER name

Specifies the owner for tables in a database when none is explicitly indicated. Also specifies the default owner of modules created by PowerHouse in ALLBASE/SQL.

Some relational databases support owners for entities such as modules or tables. If a program needs to access an entity owned by another user, you specify the owner as part of the entity name.

Defaults: username (**MPE/iX**) or logonid (**OpenVMS, UNIX, Windows**).

UNIX, Windows: The program parameters, **noshift**, **upshift**, **downshift**, and the SET statement shifting options in use may affect the case-sensitivity of the name literal.

PASSWORD string (OpenVMS, UNIX, Windows)

If the password is not included in the open-name-string of the OPEN option, the PASSWORD option specifies the password to be used to connect to the database server. Passwords are set up by the database administrator.

PowerHouse combines the open-name-string, USERID, and PASSWORD options into a valid database open specification. The separator before a password is a slash (/), which PowerHouse inserts if the password does not start with it.

UNIX, Windows: The string may be an environment variable, which must be preceded by a dollar sign ($).

Limit: This option is required if a password is not included in the OPEN open-name-string option or in an associated DATABASE resource file statement.

USERID string (OpenVMS, UNIX, Windows)

If the userid is not included in the open-name-string of the OPEN option, the USERID option specifies the userid to be used to connect to the database server. Userids are set up by the database administrator.

PowerHouse combines the open-name-string, USERID, and PASSWORD options into a valid database open specification. The separator before a userid is an at-sign (@), which PowerHouse inserts if the userid does not start with it.

UNIX, Windows: The string may be an environment variable, which must be preceded by a dollar sign ($).

Limit: This option is required if a userid is not included in the OPEN open-name-string option or in an associated DATABASE resource file statement.

Discussion

Identifying a Relational Database to PowerHouse

PowerHouse treats a relational database's catalog (or metadata) as a subdictionary of a PowerHouse dictionary. Once the relational database is identified to the PowerHouse dictionary, all PowerHouse components can access the information from the database. PowerHouse cannot define or create relational databases.

The database name in PowerHouse is the name specified in the DATABASE statement. It does not have to match the physical filespec. To use a different open name, use the OPEN option to give a file specification of a valid database. For more information, see the OPEN option on (p. 34).

Examples

MPE/iX

This example declares the ALLBASE/SQL database, ABASE, to the dictionary.

```
> DATABASE ABASE &
>   TYPE ALLBASE &
>   OPEN ABASE.TEST.PAYROLL &
>   OWNER MGR.PAYROLL
```

UNIX, Windows

This example declares the SYBASE database, NAGI, to the dictionary. The database name in PowerHouse is the name specified after the DATABASE statement. It does not have to match the physical name of the database. If it doesn't, use the OPEN option to give a valid file specification for the database.

```
> DATABASE NAGI &
>    TYPE SYBASE &
>    NULL VALUES ALLOWED &
>    DESCRIPTION &
>    "NORTH AMERICAN GEOGRAPHICAL", &
>    "INFORMATION (SYBASE DB)" &
>    OPEN "NAGI_SB@DBUSER/DBPASS"
```

OpenVMS

This example declares the Oracle Rdb database, ATLAS, to the dictionary.

```
> DATABASE ATLAS &
>    TYPE RDB &
>    DESCRIPTION &
>    "NORTH AMERICAN GEOGRAPHICAL INFORMATION" &
>    OPEN ATLAS.RDB &
>       NULL VALUES ALLOWED
```

LOW and HIGH Options

When a user specifies generic search criteria, PowerHouse uses fill characters in the BETWEEN clauses of queries.

```
>  SQL DECLARE EMPLIST CURSOR FOR &
>  SELECT EMPLOYEE, FIRST_NAME, LAST_NAME, &
>      BRANCHES.BRANCH, BRANCH_NAME &
>      FROM EMPLOYEES, BRANCHES &
>      WHERE EMPLOYEES.BRANCH = BRANCHES.BRANCH
>  ACCESS EMPLIST
>  CHOOSE FIRST_NAME PARM PROMPT "Enter a name: "
.
.
.
```

If the user enters K@ as the generic search criteria in the preceding example, the following BETWEEN clause of the SQL query is built using low and high fill characters:

```
BETWEEN 'Kll' AND 'Khh'
```

where ll is the low fill character and hh is the high fill character. The default fill characters depend on the type of database.

If the search criteria is K@@, the BETWEEN clause is

```
BETWEEN 'Kll' AND 'hh'
```

DB2: Determining the High and Low Fill Characters

The DB2 database manager compares character data using a collating sequence. The DB2 database manager allows databases to use custom collating sequences that are specified at database creation time. Once the database has been created, the collation sequence cannot be changed.

In DB2, the order in which data in a database is sorted depends on the collating sequence defined for the database. If the ASCII code page is used as the default collation sequence, the default PowerHouse values for the high and low fill characters are correct and no changes have to be made.

However, if another collating sequence is used for the database, you have to define your own high and low fill characters in your dictionary file.

With DB2, collating sequence sort order depends on the collating sequence defined for the database. The database might be an ASCII database with code page 850, or an EBCDIC database with code page 500.

For example, consider the relative collation of four characters in a EBCDIC code page 500 database, when they are collated in binary:

Character Code Page 500 Code Point

Character	Value
a	Hex ’81’
b	Hex ’82’
A	Hex ’C1’
B	Hex ’C2’

The code page 500 binary collation sequence, the desired sequence, is:

'a' < 'b' < 'A' < 'B'

If you create the database with ASCII code page 850, binary collation would yield:

Character Code Page 850 Code Point

Character	Value
a	Hex ’61’
b	Hex ’62’
A	Hex ’41’
B	Hex ’42’

The code page 850 binary collation, which is not the desired sequence, is:

'A' < 'B' < 'a' < 'b'

So the above EBCDIC-based sort and ASCII-based sort is different.

Usually, the PowerHouse collation sequence is the same as the collation sequence in the database. However, the two collation sequences can be different.

As an English language example, suppose you created your PowerHouse application with code page 850, but you want the character strings to be collated as if the data actually resides in an EBCDIC database with code page 500. The high and low fill characters you would set in your dictionary file are determined by the database collation sequence.

In the code page 500 table, ‘a’ (hex’ 81’) is the lower value character and ‘9’ (hex ’F9’) is the high value character for English language sort.

In the code page 850 table, you need to find the positions and hex values of low and high fill characters defined in the code page 500 table. For example, letter ‘a’ position and value is Hex’61’ and ‘9’ position and value is Hex’39’. Therefore, you need to set Hex’61’ as low fill string and set Hex’39’ as high fill string in your dictionary file as though ‘a’ is not the low value character in the code page 850 and ‘9’ is not the high value character code page 850.

Character	Code Page 500 Hexadecimal Value	Code Page 850 Hexadecimal Value
a	81	61
9	F9	39

To see the complete database collation sequence tables, refer to your DB2 documentation set.

DESCRIPTION

Provides a description of a dictionary entity.

Syntax

DESCRIPTION OF entity name string [[,] string]...

entity

Specifies the type of entity that's being described. This entity must be one of the following: ASC, DSC, ELEMENT, FILE, RECORD, or USAGE.

Limit: DSC is valid only in PHDPDL. Using DSC syntax in PDL generates a warning that the syntax will be ignored.

name

Names the entity being described.

string

A series of letters, numbers, or special characters enclosed in single or double quotation marks.

Limit: 60 characters per string.

Discussion

The DESCRIPTION statement gives an entity a description in the form of strings of text. The description serves as documentation for the entity.

The description can be seen in QSHOW reports. QUICK displays the description of an element when the user enters the extended help command (??) in a field that references the element.

You can use either this statement or the DESCRIPTION option of the entity you want to describe. It's an error to specify a description for an entity that already has one.

Examples

This example shows how to use the DESCRIPTION statement to add a description to an element

```
> ELEMENT ACCOUNTNUMBER &
>      USAGE NUMERIC-ID &
>      SIZE 4 &
>      HEADING "Account^Number"
.
.
.
> DESCRIPTION OF ELEMENT ACCOUNTNUMBER &
>      "A four digit account number"
```

The DESCRIPTION option of the ASC, DATABASE, DSC, ELEMENT, FILE, RECORD, and USAGE statements offers the same function:

```
> ELEMENT ACCOUNTNUMBER &
>      USAGE NUMERICID &
>      SIZE 4 &
>      HEADING "Account^Number" &
>      DESCRIPTION "A four digit account number"
```

DICTIONARY SECURITY CLASS (OpenVMS)

Names a dictionary security class (DSC) and assigns members to the DSC.

Limit: DSC is valid only in PHDPDL. DSC is parsed in PDL but generates a warning.

Syntax

**DICTIONARY SECURITY CLASS name TYPE dsc-type
option[option]...**

DSC name TYPE dsc-type option[option]...

name

Names a new DSC or one that has already been defined.

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters that are defined in the dictionary.

PowerHouse dictionaries created with the PRELOADED option are preloaded with empty classes named DICTIONARY-MANAGER, APPLICATION-MANAGER, and DICTIONARY-USER.

dsc-type

Specifies the type of dictionary access allowed to this DSC. The type must be one of

- DICTIONARY MANAGER or DMGR
- APPLICATION MANAGER or AMGR
- DICTIONARY USER or USER

Default: USER

Options

The options are DESCRIPTION, LOGONID, PASSWORD, PORTID, and UIC.

DESCRIPTION string [[,] string]...

The strings contain a description of the DSC. The description appears on the appropriate Description Screen in the PhD Screen System and when the user types SHOW SECURITY in QSHOW (if the user's dictionary security class is APPLICATION or DICTIONARY MANAGER).

Limit: 60 characters per string.

LOGONID logonid [[,] logonid]...

Assigns user names that are potential members of this dictionary security class (DSC). A logonid is an OpenVMS user name.

When a user enters a SET DICTIONARY statement in PDL or a SET SECURITY statement in QSHOW, PowerHouse checks the ID method in the DICTIONARY SECURITY ID METHOD option of the SYSTEM OPTIONS statement. The user becomes a member of the DSC if the ID method is LOGONID, and if the user's logon ID is one of the logonids. Otherwise, the user becomes a member of the default DSC, USER.

Limit: 12 characters. Logonids must be enclosed in quotation marks if they begin with a numeric.

PASSWORD password [[,] password]...

Assigns passwords that users must enter to identify themselves as a member of this DSC. A password is a 31-character string, with or without quotation marks.

When a user enters one of the following:

- a SET DICTIONARY statement in PDL
- a SECURITY statement in PDL
- a SET SECURITY statement in QSHOW,

PowerHouse checks the ID method in the DICTIONARY SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is PASSWORD, PowerHouse prompts the user for a password. The user becomes a member of the DSC if the user enters one of the passwords listed after the PASSWORD option of DSC statement. Otherwise, the user becomes a member of the default DSC, USER.

Limit: 64 characters. Enclose the password in quotation marks if it contains characters other than letters, digits, or the special name characters defined in the dictionary by the SPECIAL NAME CHARACTERS option of the SYSTEM OPTIONS statement.

PORTID portid [[,] portid]...

Assigns port IDs. A portid is an OpenVMS device name. A user who accesses PowerHouse through one of the port IDs listed may become a member of this DSC.

When a user enters a SET DICTIONARY statement in PDL or a SET SECURITY statement in QSHOW, PowerHouse checks the ID method in the DICTIONARY SECURITY ID METHOD option of the SYSTEM OPTIONS statement. The user becomes a member of the DSC if the ID method is PORTID, and if the user's port ID is one of the listed portids. Otherwise, the user becomes a member of the default DSC, USER.

Do not use this option when terminals are connected via a line multiplexer.

Limits: 16 characters. The port IDs can be listed either with or without quotation marks around each one.

UIC [gid,mid] [[,] [gid,mid]]...

Lists members of the DSC by their UICs. The UIC is defined by a gid (group id number) and a mid (member id number). The gid can be from 0-37776 (octal) and the mid can be from 0-177776 (octal). The gid and mid must be declared as a pair. The brackets around [gid,mid] are required syntax.

When a user enters a SET DICTIONARY statement in PDL or a SET SECURITY statement in QSHOW, PowerHouse checks the ID method in the DICTIONARY SECURITY CLASS ID METHOD option of the SYSTEM OPTIONS statement. The user becomes a member of the DSC if the ID method is UIC, and if the user's UIC is one of the listed UICs. Otherwise, the user becomes a member of the default DSC, USER.

Discussion

PDL uses a user's DSC and the DSC's type to determine the user's access rights to the dictionary.

All users who aren't members of a specific dictionary security class become members of the class USER.

In the PhD Screen System, Application Managers and members of the USER class can modify the dictionary as specified below. In PHDPDL, only Dictionary Managers and Dictionary Owners can modify the dictionary.

A user can only use these PDL statements:

CANCEL	DATABASE[1]	ELEMENT
EXIT	FILE[1]	INDEX
ITEM	LOAD	RECORD
REVISE	SAVE	SEGMENT
SET	SHOW DICTIONARY	USE

[1]The database and the file must already exist in the dictionary.

In addition to all the statements a user can use, the application manager can use the following PDL statements:

ASC	DATABASE	ETP (Execution-Time Parameters)
FILE	PERMIT	SYSTEM OPTIONS[1]
TRANSACTION	USAGE	USER MODE

[1]Except DSC ID METHOD and DICTIONARY OWNER

The dictionary manager can use all PDL statements.

The dictionary owner always has dictionary manager rights to the dictionary. The SHOW SYSTEM statement in QSHOW displays the logon ID of the dictionary owner.

Example

Suppose your dictionary contains the following statements:

```
> SYSTEM OPTIONS &
>      DSC ID METHOD PASSWORD

> DICTIONARY SECURITY CLASS DBA &
> TYPE DICTIONARY MANAGER &
> PASSWORD "dev"
> LOGONID MANAGER

> DICTIONARY SECURITY CLASS DEVELOPMENT &
> TYPE APPLICATION MANAGER &
> PASSWORD "apps" &
>      LOGONID PROGRAMMER
```

If you start QSHOW, the PhD Screen System, or PHDPDL with this dictionary, the component prompts you for a password. If you enter "dev", you have dictionary manager privileges. If you enter "apps", you have application manager privileges. If you enter anything else, you have user privileges.

If instead of the SYSTEM OPTIONS statement in the previous example, the dictionary contained

```
> SYSTEM OPTIONS & DSC ID METHOD LOGONID
```

then the user MANAGER has dictionary manager privileges. User PROGRAMMER has application manager privileges. All other users have user privileges.

ELEMENT

Declares an element.

Syntax

ELEMENT name element-type [SIZE] n [option]...

ELEMENT name USAGE [IS] usage [SIZE n [BYTES]] [option]...

name

A unique name that's meaningful to system users, designers, and programmers. The name is used for all references to the element in PowerHouse.

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters that are defined in the dictionary.

element-type

The element type must be one of the following:

- CHARACTER or C
- DATE or D
- NUMERIC or N

[SIZE] n

Specifies the maximum number of characters or digits allowed for values of the element.

If you use the first form of the ELEMENT statement (without a usage), you must enter a number for the element size; the keyword SIZE is optional.

If you use the second form of the ELEMENT statement (with USAGE), the size is usually determined by the usage and you can't change it. However, there are some usages that don't set the element size (they show an element size of 0). If you use one of these usages, then you must enter both the keyword, SIZE, and a number for the size as part of the ELEMENT statement.

PowerHouse uses element sizes to determine default pictures and item sizes. The number of substitution characters (^) in the picture, the item datatype and size, and the allowed values determine the actual number of characters that can be entered or displayed for a value.

The size of a date element must be one of 6, 8, 10, 12, 14, 15, or 16.

- a size of 8 means that the date includes a two-digit century
- a size of 6 means that the century is excluded
- a size of more than 8 means date, century and time are included

Date and Time Support

The SIZE option controls the time portion to be formatted and displayed. It also determines the default datatype. The following table shows the different time formats that are available based on the DATE SIZE option. Assume the date format is YYYYMMDD and the separator is a slash (/):

Size	Display Format	Example
8	YYYY/MM/DD	2001/12/29
10	YYYY/MM/DD HH	2001/12/29 23
12	YYYY/MM/DD HH:MM	2001/12/29 23:59
14	YYYY/MM/DD HH:MM:SS	2001/12/29 23:59:59
15	YYYY/MM/DD HH:MM:SS:N	2001/12/29 23:59:59 9

Size	Display Format	Example
16	YYYY/MM/DD HH:MM:SS:NN	2001/12/29 23:59:59 99

Any value greater than eight in the ELEMENT DATE SIZE statement assumes that the datatype is DATETIME. All other datatypes result in an error. A DATETIME item always has the century included, so the date portion of the item is considered to be a full date.

Limit: 2047 for character elements; up to 31 for numeric elements. Numeric elements are also limited by the datatype.

USAGE [IS] usage

Identifies the usage on which the element is to be based.

The usage must have been defined previously. It can be one of the predefined usages that are supplied with the PowerHouse dictionary, or it can be a usage that has previously been defined by the USAGE statement. For information about the predefined usages that are available, see Chapter 3, "Standard Elements and Usages".

Since usages determine the value of most element attributes, you can only use the following attributes in combination with a usage:

- default item datatype
- item size (in bytes)
- number of occurrences
- default column heading for QUIZ reports
- default item label
- default help message for the item
- element description

Options

ELEMENT Options		
ALLOW|NOALLOW CENTURY	BWZ	DECIMAL
[DEFAULT][ITEM] DATATYPE	DESCRIPTION	DOWNSHIFT|UPSHIFT
FILL	FLOAT	FORCE|NOFORCE CENTURY
FORMAT	HEADING	HELP
INITIAL	INPUT SCALE	LABEL
LEADING SIGN	NULLSEPARATOR| NONULLSEPARATOR	OCCURS
OUTPUT SCALE	PATTERN	PICTURE
SEPARATOR	SIGNIFICANCE	TRAILING SIGN
VALUES		

ALLOW|NOALLOW CENTURY

ALLOW CENTURY specifies that the user can enter a century on date fields even though only a two-digit year is specified in the date format. The option applies to century-included date fields with a two-digit year format.

When ALLOW CENTURY is specified, date fields become horizontal scrolling fields. The user can then enter the date, including the century, in the same space as the date without the century. The century is not displayed after input.

Limit: Valid only for century-included dates.

Default: The ELEMENT default depends on what is specified for the same option on a related USAGE statement or the SYSTEM OPTIONS statement in PDL.

BWZ

Means blank when zero. When this option is in effect, PowerHouse displays blanks instead of zeros for numeric item values that equal zero. This option is useful for elements such as phone numbers or customer numbers, whose values are not used in calculations. For example, you don't usually want to display the value "000-0000" for a North American telephone number. If you don't include the BWZ option for a numeric item, then zero values are displayed as 0.

Limit: Valid only for numeric elements that don't have a usage.

Default: 0

DECIMAL n

Specifies the number of decimal positions in a numeric element. PowerHouse uses this option to determine defaults for the input scale, significance, and picture.

[DEFAULT] [ITEM] DATATYPE type [SIGNED|UNSIGNED] [SIZE n [BYTES]]

Specifies the default item datatype, size, and number of occurrences. DEFAULT and ITEM are for documentation only.

The values specified here are used if these attributes are not specified in the ITEM statement. If nothing is specified here or in the ITEM statement, PowerHouse provides defaults based on the element type and size.

DATATYPE type

Specifies how item values are stored. The PowerHouse datatypes are:

Datatype	Item Type[1]	Stores
CHARACTER	C	character items in one character per byte
DATETIME	D	10, 12, 14, 15 or 16 digit datetimes
FLOAT	D[2], N	numeric items as standard floating point numbers
FREEFORM	D[2], N	numeric items as a series of characters
G_FLOAT (**OpenVMS**)	D[2], N	numeric items as eight-byte floating point numbers
INTEGER	D[2], N	numeric items as binary numbers
INTERVAL	N	difference between two datetimes
JDATE	D	dates as the Julian date
PACKED	D[2], N	numeric items as packed decimal numbers
PHDATE	D	dates as PowerHouse dates
VARCHAR	C	one or more characters preceded by an integer representing the current string length
VMSDATE (**OpenVMS**)	D	the standard eight-byte (64-bit) OpenVMS date
ZDATE	D	dates in vendor-specific format
ZONED	D[2],N	numeric items as zoned decimal numbers

Datatype	Item Type[1]	Stores

[1]Key: C, Character; D, Date; N, Numeric

[2]Note that FLOAT, FREEFORM, INTEGER, PACKED, and ZONED may be defined as the item datatype for date type elements in PDL.

SIGNED|UNSIGNED

Indicates whether the datatypes INTEGER, PACKED, and ZONED are SIGNED or UNSIGNED. INTEGER SIGNED can store negative values; INTEGER UNSIGNED cannot. ZONED and PACKED datatypes can store positive or negative numbers, regardless of whether SIGNED or UNSIGNED is specified.

Limit: Valid only for datatypes INTEGER, PACKED, and ZONED. The SIGNED and UNSIGNED options are mutually exclusive and must immediately follow the datatype.

Default: SIGNED for datatypes INTEGER and PACKED; UNSIGNED for ZONED.

SIZE n [BYTES]

Specifies the storage size, in bytes, of items based on this element.

DESCRIPTION string [[,]string]...

Specifies a description that serves as documentation about the element. The description can be seen in QSHOW reports. QUICK screen users can also see the description when they enter two question marks (??) in a field that references the element.

Limit: 60 characters per string.

DOWNSHIFT|UPSHIFT

UPSHIFT causes QUICK to change the user's input to uppercase for this element. DOWNSHIFT causes QUICK to change the user's input to lowercase. If you specify neither, QUICK leaves the input in the case that the user enters, and treats uppercase and lowercase letters as distinct.

Limit: Valid only for character elements that don't have a usage.

FILL char

Specifies the character that's used to fill any unused space to the left of the most significant digit, float character, or leading sign in the picture. The fill character also replaces unnecessary leading nonsubstitution characters (including commas and leading spaces).

For example, the following attributes work together to display a dollar amount that is preceded by leading asterisks:

Fill	Float	Picture	Value	Display
*	$	"^,^^^,^^^.^^"	123456	***$1,23.56

Limit: Valid only for numeric elements that don't have a usage.

FLOAT char

Specifies the float character that's inserted immediately to the left of the most significant digit.

Add either a space or an extra picture substitution character (^) to the left side of the picture to make room for the float character, or else the field may overflow.

For example, the following element attributes work together to display a dollar sign ($) in front of an item value:

Stored value	Picture	Float	Displayed value
1234	"^^^^^"	$	$1234
56789	"^^^.^^"	$	$567.89

Limit: Valid only for numeric elements that don't have a usage.

FORCE|NOFORCE CENTURY

FORCE CENTURY specifies that the user must enter a century on all century-included date fields. The option applies to century-included dates with two or four-digit year formats.

Limit: Valid only for century-included dates.

Default: The ELEMENT default depends on what is specified for the same option on a related USAGE statement or the SYSTEM OPTIONS statement in PDL.

FORMAT date-format

Specifies the format for entering and displaying date item values. Date values can be entered either with or without separator characters. A date-format can be one of the following:

Date-format	Example	Date-format	Example
YYMMDD	01/05/23	YYMMMDD	01/MAY/23
YYYYMMDD	2001/05/23	YYYYMMMDD	2001/MAY/23
YYMM	01/05	YYMMM	01/MAY
YYYYMM	2001/05	YYYYMMM	2001/MAY
YYDDD	01/125	YYYYDDD	2001/125
MMDDYY	05/23/01	MMMDDYY	MAY/23/01
MMDDYYYY	05/23/2001	MMMDDYYYY	MAY/23/2001
MMYY	05/01	MMMYY	MAY/01
MMYYYY	05/2001	MMMYYYY	MAY/2001
MMDD	05/23	MMMDD	MAY/23
DDMMYY	23/05/01	DDMMMYY	23/MAY/01
DDMMYYYY	23/05/2001	DDMMMYYYY	23/MAY/2001
DDMM	23/05	DDMMM	23/MAY
DDDYY	125/01	DDDYYYY	125/2001

Date-format	Example	Date-format	Example
YYYY - four digit year (e.g., 2001)			
MM - two digit month (e.g., 05)			
MMM - three character month name (e.g., MAY)			
DD - two digit day for a month (e.g., 23)			
DDD - three digit day for a year (e.g., 365)			
Regardless of the output order of the date, the internal working format is YYMMDD (for dates without centuries), YYYYMMDD (for dates with (for Julian dates)			

If the FORMAT option is used but the SEPARATOR option isn't, the only separator character that QUIZ accepts is the separator character specified by System Options, or if it isn't specified, a slash (/).

Single-digit day and month entries are accepted if the user enters the separator character, as in 4/8/2001. An entry of 4AUG2001 is also allowed, because PowerHouse accepts a single-digit day entry if the middle value is a three-character month.

If a two-digit year is specified in the date format, applications won't accept a four-digit year. A two-digit year is represented by YY (for example, 01).

A three-digit day of the year from 1 to 366 is represented by DDD.

If a four-digit year is specified in the date format, you can only enter a two-digit year if you enter a separator character between the year and any adjacent numeric component of the date. The default century is added automatically.

Dates can always be entered with either the MM or MMM month format. Although values for date items can be entered in a variety of formats, the values are always stored in either YYMMDD or YYYYMMDD form.

Limit: Valid only for date items. This option only affects the entry format; the display format isn't affected.

Default: YYYYMMDD for eight-digit dates; YYMMDD for six-digit dates.

HEADING string

Provides a default column heading for QUIZ reports.

A multiline heading character (^) embedded in the heading produces multi-line headings. For example, the heading "Employee^Number" appears in a QUIZ report as

```
Employee
 Number
```

If you do not specify a multiline heading, when the column heading is much longer than the column data width, PowerHouse wraps the heading automatically. PDC dictionaries determine the heading at dictionary compile time; PHD dictionaries determine the heading at component runtime. The algorithms used to determine where to split the text and the resultant column width are different for PDC and PHD dictionaries. If you are switching between dictionary types, you may have to reformat your reports or specify your own wrapped headings.

If no heading is specified, QUIZ uses the element name as a column heading.

Limit: 60 characters per string.

Default: The element name with all hyphens and underscores replaced with blanks. The first letter of each word is upshifted; the remaining letters are downshifted. Multi-line headings, to a maximum of three lines, are constructed by replacing blanks with the multiline heading character whenever a word or group of words is longer than the element display size. The first and second line in a heading is less than or equal to the display size, but the third may be longer.

HELP string

Specifies the help string that QUICK displays when the user enters the help command (?) in a field that references the element.

Limit: 60 characters per string.

Default: The element name, size, and type.

INITIAL value

Specifies the default value for items that correspond to this element. QUICK and QTP initialize items with the initial value. QUIZ uses the initial value to fill in default values in optional linkages.

Limit: The value can't have more digits than the element's element size, nor can it have more decimal places than the element's number of decimal places.

Default: Character items are initialized to spaces; numeric and date items are initialized to zero.

INPUT SCALE n

Specifies the value of the input scale. QUICK multiplies the user's input value by 10 raised to the power of the input scale before it stores the value.

Limit: -16 to 16

Default: The value specified on the DECIMAL option.

LABEL string

Specifies the default prompt when the PARM prompt option is used in the CHOOSE statement in QUIZ or QTP, and the default field label for QUICK screens.

For example, the label "Valid 6-digit Employee Number:" causes QUIZ to display the prompt

```
Valid 6-digit Employee Number:
```

if the employee number item is specified in a CHOOSE statement with a PARM option.

Limit: 60 characters per string.

Default: The element's heading. If no heading is specified, the element's name.

LEADING SIGN char

Specifies a character that's placed to the left of a value when the value is negative.

Add either a space or an extra substitution character (^) to make room for the leading sign, or else the field may overflow.

Limit: Valid only for numeric elements that don't have a usage.

Default: "-" if the DEFAULT ITEM DATATYPE is FLOAT, FREEFORM, INTEGER SIGNED, PACKED SIGNED, or ZONED SIGNED. Otherwise the default is a blank, that is, no leading sign.

NULLSEPARATOR|NONULLSEPARATOR

NULLSEPARATOR specifies that all dates are to be displayed without a separator. This allows display of century-included dates in the same space as century-excluded dates.

The DATE SEPARATOR option is used for display formatting if NULLSEPARATOR is not used, or is canceled by the NONULLSEPARATOR option.

The DATE SEPARATOR option may be used during input. If NULLSEPARATOR is specified, the value is redisplayed after formatting without the separator.

Default: The ELEMENT default depends on what is specified for the same option on a related USAGE statement or the SYSTEM OPTIONS statement in PDL.

OCCURS n

Specifies the number of times an item repeats in an array.

The value specified here is used if this attribute is not specified on the ITEM statement.

Limit: 1 to 4,096. (QUICK can handle a maximum occurrence of 255.)

Default: 1

OUTPUT SCALE n

Establishes the output scaling factor that's applied when values are displayed.

Before it's displayed, the stored value of the item is multiplied by 10 raised to the power of the output scale value.

OUTPUT SCALE is needed for floating point numbers since fractional portions of stored values are rounded for display. If the output scale is zero, all digits to the right of the decimal in a floating-point number are truncated when the number is rounded.

For example, the following item attributes work together with the output scale to display floating point values:

Stored value	Picture	Output scale	Displayed value
12.54	"^^^^.^^"	0	0.13
12.54	"^^^^.^^"	2	12.54

Limit: -16 to 16. Valid only for numeric elements that don't have a usage.

Default: 0

PATTERN string

Specifies a string of characters and metacharacters that provides a general description of values. QUICK compares the user's input with the pattern. If the input doesn't match the pattern, QUICK displays an error message.

Limit: 60 characters per string.

PICTURE string

Establishes the output picture used to format the item value for display. The string is made up of substitution characters (^), and nonsubstitution characters (any other characters).

Character items are formatted in the following way:

1. The item is processed from left to right, substituting one character from the item for each substitution character in the picture. Nonsubstitution characters remain unchanged.
2. If there are fewer substitution characters in the picture than characters in the item value, the remaining characters in the item aren't displayed.
3. If there are more substitution characters in the picture than characters in the item value, spaces are padded to the right of the item.

As an example, the item value "FHSMITH" is formatted as follows:

Picture display	Resulting
"^^^^^^^"	FHSMITH
"^^^^^"	FHSMI
"^.^. ^^^^^"	F.H. SMITH

Numeric items are formatted in the following way:

1. The item is scaled by the output scale and rounded to the nearest whole number.
2. The integer portion of the item is processed from right to left, substituting one digit from the item for each substitution character in the picture until all significant (nonzero) digits have been processed. Nonsubstitution characters remain unchanged.
3. Until the element significance is reached, leading zeros are substituted for each substitution character. Nonsubstitution characters remain unchanged.
4. The float character is added.
5. Leading and trailing signs are added for negative values.

6. The remaining portion of the picture is filled with the fill character.
7. If there isn't enough room in the picture to accommodate all of the significant digits of the item value or the leading or trailing signs, the item is filled with the overflow character (#).

For example, the value 1578 is formatted as follows:

Picture display	Resulting
"^^^^"	1578
"^^.^^"	15.78
"^^,^^^,^^^"	1,578

Limit: 60 characters per string. Valid only for numeric and character elements that don't have a usage.

Defaults: The number of substitution characters in the default picture for numeric items is the number of digits in the item plus one (to accommodate the leading sign).

For character elements, the default picture is one substitution character for each character in the element.

For numeric elements, the default consists of:

- one leading blank if a LEADING SIGN character other than blank is specified
- one leading blank if a FLOAT character other than blank is specified
- one substitution character for each digit to the left of the decimal place. The SYSTEM OPTIONS DECIMAL character is inserted every fourth position from right to left.
- the SYSTEMS OPTIONS DECIMAL character plus one substitution character for each digit to the right of the decimal place if the ELEMENT DECIMAL option is greater than 0
- one or two trailing blanks (corresponding to the characters in the sign) if a TRAILING SIGN other than blank is specified

SEPARATOR char

Specifies the character that separates the day, month, and year portions of a date element for display.

For example, the separator character "-" produces dates that are displayed like this: "99-05-25".

Limit: Valid only for date elements that don't have a usage.

Default: A slash (/) unless a different date separator is specified in the SYSTEM OPTIONS statement.

SIGNIFICANCE n

Specifies the minimum number of digits and characters displayed. The SIGNIFICANCE option forces the printing of leading nonsubstitution characters and leading zeros.

For example, if letters or special characters must always appear as part of an element picture, the significance must be large enough to force the display of all desired letters or characters. The value "1578" is displayed as follows, based on the indicated significance:

Picture	Significance	Displayed value
"^^.^^^^"	6	0.1578
"^^^^.^^"	3	15.78
"^^^^.^^"	7	0015.78
"^^.^^^^"	4	1578
" P.O.^^^^-^^"	11	P.O.0015-78

Picture	Significance	Displayed value
"^^.^^^^%"	5	1578%

Limit: 60 characters or digits.

Default: The number of decimal positions of the element, plus one if the number of decimal positions is greater than zero, plus the length of the trailing sign, plus one.

TRAILING SIGN string

Specifies a string of one or two characters to be placed to the right of a value when the value is negative.

You must provide sufficient nonsubstitution characters in the right-most portion of the picture to accommodate the trailing sign. If the picture is too small, then values based on the element will be filled with the overflow character (#). To accommodate a trailing sign, you must include one or two nonsubstitution characters (usually spaces) in the right-hand side of the element picture.

If no leading or trailing sign is specified, the entry of negative values isn't allowed in fields on QUICK screens.

To place parentheses around negative numbers, use the LEADING SIGN and TRAILING SIGN options together. For example, a leading sign of "(" with a trailing sign of ")" displays the value -123.45 as

```
(123.45)
```

A leading or trailing sign is required to store or display negative values.

For example, the number -1578 is formatted as follows:

Picture	Leading sign	Trailing sign	Resulting Display
"^^^,^^^ "	none	CR	1,578CR
"..^^^,^^^ "	(	)	(1,578)

Limit: Valid only for numeric elements that don't have a usage.

Default: Two spaces (no trailing sign).

VALUES value-set

Specifies acceptable values for an element.

QUICK uses the allowed values to validate user entries. QTP uses them to perform bulk validation of input files and to validate user-entered execution-time parameters. QUIZ also uses dictionary values to validate execution-time parameters. For more information, see the CHOOSE and EDIT statements in the *QUIZ* and *QTP Reference* books and the FIELD statement in the *QDESIGN Reference* book.

Limit: The value of the element can't have more digits than the element size, nor can it have more decimal places than the element's number of decimal places.

You can specify descriptive captions that are used when the corresponding item is referenced by a field with a selection box; the captions are displayed in the selection box in place of the actual entry values for the field. The value-set includes the CAPTION option, as in

**value [CAPTION string][TO value [CAPTION string]]
  [[,] value [CAPTION string][TO value [CAPTION string]]]...**

Selection boxes may be unsuitable for fields that accept ranges of values (using value TO value syntax), because they display only the first and last values of each range.

CAPTION string

Displays descriptive captions in selection boxes in place of the actual entry values for the field. When the user chooses a caption in the selection box, the corresponding value is entered and displayed in the field.

Limits: The maximum size of the caption string is 60 characters. Captions are used only in selection boxes; if no selection box is specified, then the captions are ignored.

Discussion

The ELEMENT statement defines an element in the PowerHouse dictionary. An element is the smallest category of data in a PowerHouse application.

One form of the ELEMENT statement includes a name, a type, and a size. The other form includes a name and a usage. All other attributes are optional.

Examples

Using the Element Statement

The following example demonstrates the use of multiple options on the ELEMENT statement in a dictionary design.

```
> ELEMENT DISCDAYS NUMERIC SIZE 3 BWZ
> ELEMENT DISCPCT NUMERIC SIZE 4                &
>    DECIMAL 2 INPUT SCALE -2                   &
>    SIGNIFICANCE 5                             &
>    OUTPUT SCALE 4 PICTURE "^^.^^%"            &
>    HEADING "VENDOR^DISCOUNT^(%)"              &
>    LABEL "DISCOUNT %"                         &
>    HELP "THE VENDOR'S DISCOUNT AS A PERCENTAGE" &
>    DESCRIPTION "THIS FIELD CONTAINS"          &
>                "VENDOR'S DISCOUNT PERCENTAGE"   &
>                "THAT IS APPLIED TO ALL"         &
>                "PURCHASES FROM THE VENDOR"      &
>                "AND IS USED TO CALCULATE THE"   &
>                "INVENTORY ITEM PRICE"
```

These statements are taken from a larger design for an inventory system.

In the next example, the three ELEMENT statements describe the display that's generated for the elements TRANSNO, TRANSTOTAL, and TOTALQTY.

- TRANSNO describes a transaction number that displays a maximum of seven digits.
- PICTURE illustrates the display format.
- SIGNIFICANCE specifies a maximum of eight digits, forcing the display of leading zeros in associated items.
- TRANSTOTAL displays a maximum of ten digits with two decimal positions. PICTURE and FLOAT display a dollar amount for this element's corresponding items.
- LEADING SIGN indicates that this amount can have a negative value.

```
> ELEMENT TRANSNO
>      NUMERIC SIZE 7 &
>      PICTURE "^^^-^^^^" &
>      SIGNIFICANCE 8
>
>
>
>
>
> ELEMENT TRANSTOTAL NUMERIC SIZE 10 DECIMAL 2 &
>      PICTURE "  ^^,^^^,^^^.^^" &
>      FLOAT "$" LEADING SIGN "-"
>
>
> ELEMENT TOTALQTY NUMERIC SIZE 8 &
>      PICTURE " ^^^^^^^^" &
>      LEADING SIGN "-"
```

In many systems, attributes of one element are often the attributes for other elements. For example, the attributes assigned to the element TRANSNO could be the same for a number of other elements in this dictionary. Instead of defining the same attributes for a number of different elements, you could create a usage describing an element of this type:

```
> USAGE IS INVENID &
>      NUMERIC SIZE 7 &
>      PICTURE "^^^-^^^^" &
>      SIGNIFICANCE 8
```

Once the usage is defined, the ELEMENT statement for TRANSNO is

```
> ELEMENT TRANSNO &
>      USAGE INVENID
```

Using the BWZ Option

The following example illustrates the use of the BWZ option. In this example:

- BUYERNO isn't displayed unless a non-zero value for the application user is entered.
- BWZ displays a space if the value for this element's corresponding item is zero.

```
> ELEMENT BUYERNO &
>      NUMERIC SIZE 2 &
>      SIGNIFICANCE 3 &
>      BWZ &
>      PICTURE "B^^"
```

Using the INPUT SCALE and OUTPUT SCALE Options

The INPUT SCALE and OUTPUT SCALE options in the following example display the corresponding item at its entered value multiplied by 105. An input scale of zero stores the value of the item in its entered state. (If the item is stored as an integer, only the integer portion of the value is stored.) This element can ultimately be used for an item that acts as a method of quickly determining the retail value of a piece of merchandise.

```
> ELEMENT CONVUNIT NUMERIC SIZE 8 DECIMAL 5 &
>    INPUT SCALE 0 OUTPUT SCALE 5 &
>    DESCRIPTION "CONVERSION FACTOR BETWEEN UNIT" &
>    "WHOLESALE AND UNIT RETAIL."
```

Using the DATE SIZE Option

The following PDL code is an example of date support. In the following example:

- ITEM ONE is correctly specified.
- ITEM TWO defaults to datatype DATETIME.
- ITEM THREE defaults to datatype PHDATE.
- ITEM FOUR defaults to INTEGER SIZE 4.

```
> ELEMENT ONE DATE SIZE 15 FORMAT YYYYMM
> ELEMENT TWO DATE SIZE 16 FORMAT MMDDYYYY
> ELEMENT THREE DATE SIZE 6
> ELEMENT FOUR DATE SIZE 8
> ELEMENT FIVE DATE SIZE 11
*E* Invalid date element/usage size.
> ELEMENT SIX DATE SIZE 10
.
.
.
> ITEM ONE DATATYPE DATETIME
> ITEM TWO
> ITEM THREE
> ITEM FOUR
> ITEM SIX DATATYPE PHDATE
*E* THE SPECIFIED DATA TYPE IS INVALID FOR THIS ELEMENT TYPE.
```

Using the CAPTION Option

In the following example, the LANGUAGE element includes the captions "English", "French", and "German" for the entry values "E", "F", and "G", respectively. If the corresponding item is referenced by a field with a selection box, then the selection box displays the captions in place of the entry values:

```
> ELEMENT LANGUAGE  CHARACTER SIZE 1  &
>         VALUE "E" CAPTION "English" &
>         VALUE "F" CAPTION "French"  &
>         VALUE "G" CAPTION "German"
```

END STRUCTURE

Ends an item substructure.

Syntax

END STRUCTURE

Discussion

The END STRUCTURE statement marks the end of a set of ITEM statements that define a substructure. The BEGIN STRUCTURE statement marks the beginning of the substructure. The ITEM statements form a substructure of the item that was most recently defined before the BEGIN STRUCTURE statement.

For more information the BEGIN STRUCTURE statement, see (p. 28).

Example

For an example of the END STRUCTURE statement, see (p. 28).

EXECUTION TIME PARAMETERS (OpenVMS)

Defines a default location for data files.

Syntax

**EXECUTION TIME PARAMETERS usermode [DEFAULT]
LOCATION location**

ETP usermode [DEFAULT] LOCATION location

usermode

The name of a user mode that has been defined in the dictionary.

DEFAULT

An optional word, used only for documentation.

LOCATION

Indicates the parts of an OpenVMS file specification that PowerHouse can use in combination with file open names to find files in specific locations. The location is not used to locate subfiles.

Discussion

The ETP statement assigns a file location to a group of PowerHouse users who make up a user mode. Use the USER MODE statement to declare each user mode and list its members or to reference a user mode that already exists in the dictionary. PowerHouse uses the file location at execution time to determine the complete file specification for accessing application data.

Security and Compiled Applications

The following information is compiled into QUIZ reports, QTP runs, and QUICK screens:

- ASC ID Method
- the name of the ASC and the association of the ASC with records and elements
- User Modes and locations specified in the ETP statement

If any of this information changes, the corresponding screens, reports, and runs need to be recompiled.

The membership of ASCs and User Modes is dynamic. You can add IDs to and delete IDs from ASCs without requiring that screens, reports, and runs be recompiled.

Examples

The file open name declared in the dictionary combines with the file location to create a complete file specification. For example,

Open Name:	MYFILE.DAT
File Location:	[INVENTORY.TEST]
File Specification:	[INVENTORY.TEST]MYFILE.DAT

The location doesn't override any part of the file specification given in the open name. For example,

Open Name:	[INVENTORY.PROD]MYFILE.DAT
File Location:	[INVENTORY.TEST]

File Specification:	[INVENTORY.PROD]MYFILE.DAT

Suppose your dictionary contains the following statements:

```
> SYSTEM OPTIONS
>      ASC ID METHOD PASSWORD
>
>USER MODE DEVELOPMENT &
>      PASSWORD "DEV" &
>      LOGONID TEST
>
> USER MODE PRODUCTION
>      PASSWORD "prod" &
>      LOGONID PROD
.
.
.
> EXECUTION TIME PARAMETERS DEVELOPMENT &
> LOCATION TEST:
> EXECUTION TIME PARAMETERS PRODUCTIONS &
> LOCATION PROD:
```

If you start a PowerHouse component with this dictionary, the component prompts you for a password. If you enter "dev", you access data files pointed to by TEST. If you enter "prod", you access data files pointed to by PROD.

If, instead of the SYSTEM OPTIONS statement in the previous example, the dictionary contains

```
> SYSTEM OPTIONS
>      ASC ID METHOD LOGONID
```

then user, "test", accesses data files pointed to by TEST, and user, "prod", accesses data files pointed to by PROD.

EXIT

Ends a PDL and PHDPDL session.

Syntax

EXIT

Discussion

The EXIT statement ends your session and returns control to the operating system or invoking program.

EXIT and QUIT are interchangeable.

FILE

Defines a file and its attributes or identifies a relational database.

Syntax

**FILE name ORGANIZATION file-organization [TYPE type]
 [option]...**

name

Names a unique PowerHouse file name (not an operating system file name). It can be given to a database to identify it to the PowerHouse dictionary. The name does not have to be the same as the file's or database's physical name (see the OPEN option on (p. 63)).

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters that are defined in the dictionary.

ORGANIZATION file-organization

Determines the type of file access allowed. PowerHouse supports the following types of file organizations:

DATABASE (MPE/iX, UNIX, Windows)

On MPE/iX, specifies that the file is an IMAGE database ("TYPE IMAGE").

On UNIX and Windows, specifies that the file is an Eloquence database ("TYPE ELOQUENCE").

DIRECT

Specifies that records can be read sequentially or by record number. The first record is 0 (**MPE/iX, UNIX, Windows**) or 1 (**OpenVMS**).

New records are added at the end of the file. Existing records can be updated, but not deleted.

INDEXED

Specifies that records can be read sequentially or by index value. Records can be added, updated, and deleted.

Limit: An indexed file must have at least one INDEX statement.

RELATIONAL

Specifies that the file is a relational database.

RELATIVE (MPE/iX, OpenVMS)

The same as DIRECT, except that records can also be deleted. New records can be added in a specific location by a record number. Deleting records leaves "gaps" in the file.

SEQUENTIAL

Specifies that records can only be read sequentially. New records are added to the end of the file. Existing records can't be updated or deleted.

For information about the relationship between ORGANIZATION and TYPE, see (p. 67).

TYPE type

PowerHouse supports the following file types:

Platform	Supported File Type
MPE/iX:	ALLBASE, IMAGE, KSAM, KSAMXL, KSAM64, MPE

Platform	Supported File Type
OpenVMS:	MBX, ORACLE, RDB, RDB/VMS, RMS
UNIX:	C-ISAM, SYBASE, ORACLE, DB2, ODBC, ELOQUENCE
Windows:	DOS, UNIXIO, DISAM, DB2, ODBC, ORACLE, SYBASE, ELOQUENCE

KSAM, KSAMXL, KSAM64 (MPE/iX)

KSAMXL file type specifies a Native Mode KSAM file system. These files allow re-use of deleted records.

PowerHouse considers KSAM, KSAMXL, and KSAM64 files to be identical with respect to file access from QUICK, QUIZ, and QTP.

ODBC (UNIX, Windows)

Windows: For MS SQL Server, use ODBC. **UNIX:** PowerHouse supports the DataDirect ODBC connection to Microsoft SQL Server.

For information about the relationship between ORGANIZATION and TYPE, see (p. 67).

Options

FILE Options	
`ASCII|BINARY`	`BLOCKING FACTOR`
`CAPACITY`	`CHRONOLOGICAL|NOCHRONOLOGICAL`
`CREATE|NOCREATE`	`[NO]CRITICAL ITEM UPDATE`
`DEFAULT DATE [IS] NULL`	`DESCRIPTION`
`HIGH [FILL] [HEXADECIMAL] string`	`KEYFILE`
`LOW [FILL] [HEXADECIMAL] string`	`NULL VALUES [NOT] ALLOWED`
`OPEN`	`OWNER`
`PASSWORD`	`RECORD FORMAT`
`REUSE`	`SCOPE`
`SUPERSEDE`	`USERID`

ASCII|BINARY (MPE/iX)

ASCII indicates that short records are padded with blanks and the record sizes are in bytes. BINARY pads short records with zeros, but the record sizes are in two-byte words. This is important when the data is accessed with other software tools.

Limit: Valid only with direct, indexed, relative, and sequential files.

Default: ASCII

BLOCKING FACTOR n (MPE/iX)

Sets the number of records per block when QUTIL creates a file.

Limit: Valid only with direct, indexed, relative, and sequential files.

CAPACITY n (MPE/iX)

Sets the number of records allotted by QUTIL when creating a file.

Limit: Valid only with direct, indexed, relative, and sequential files.

Default: 1,023 records

CAPACITY n (OpenVMS)

For relative files, capacity specifies the maximum number of records allowed. RMS will not expand the file past the specified number of records.

The capacity is also used to determine the initial file size for RMS indexed, sequential, and direct files. The initial file size is set to (capacity X record size)/512=number of blocks.

For direct, indexed, and sequential files, and for relative files without a specified capacity, RMS automatically expands the file to accommodate new records within your system's limits. PowerHouse does not assign a default capacity.

For MBX files, the capacity specifies the size of the mailbox, which limits the number of messages that can be stored.

Limit: Not available for relational databases.

CHRONOLOGICAL|NOCHRONOLOGICAL (UNIX, Windows)

The CHRONOLOGICAL option states that values for a repeating index are stored chronologically in their order of entry. If you specify the NOCHRONOLOGICAL option, values are stored randomly, using any available space in the file.

CREATE|NOCREATE

Notifies QUTIL to create a file when the CREATE ALL statement is specified, and not to create a file when NOCREATE is specified.

The CREATE|NOCREATE options are overridden when QUTIL CREATE is used to explicitly specify the file.

When two or more files of the same file type use the same open name, this option designates which file is to be declared to the file management system when the file is created. QUTIL creates the first file in alphabetical order which does not have the NOCREATE option. The specified file determines the record length, number of keys, key positions, and other key attributes when the file is created.

OpenVMS: When using the PhD Screen System to maintain your dictionary, NOCREATE does not allow the file to be created with the CREATE Action field command on the PhD Record Screen. Instead, the File Maintenance Screen must be used to create the file. This option enables application and dictionary managers to prevent general dictionary users from accidentally losing data by recreating existing files.

Limit: Not available for relational files.

Default: CREATE

[NO]CRITICAL ITEM UPDATE (MPE/iX, UNIX, Windows)

When critical item update is enabled for a database, IMAGE and Eloquence allow detail dataset key items to be updated in place using DBUPDATE instead of deleting the record and then adding it. Critical item update is used only with detail dataset records.

Critical item update capability must be specified in PowerHouse and enabled externally by the database administrator. When critical item update is enabled for both the dictionary and the database, all data updates performed on detail datasets will update in place. If critical item update is enabled only for the database, or only for PowerHouse, then the item is deleted and added using DBDELETE and DBPUT.

The use of this option does not change the chronological order of the affected data.

The use of this option results in significantly faster updates of detail datasets. A critical item is a search item (also known in PowerHouse as a Key item) or sort item for the detail data set.

Limit: Valid only for IMAGE files (MPE/iX) and Eloquence files (UNIX, WIndows).

Default: NO CRITICAL ITEM UPDATE

DEFAULT DATE [IS] NULL

Enables NULL support for date fields in relational databases. Specifies that when a user enters a value of 0 in a date field, the date column is set to null. Setting the value to null lets users enter dates with a value of 0, but prevents PowerHouse from storing the date as 0 which is an invalid value in the database. Dates that are read as null are automatically converted to 0.

DESCRIPTION string [[,]string]...

Specifies a description that serves as documentation about a file. The description can be seen in QSHOW reports.

Limit: 60 characters per string.

HIGH [FILL] [HEXADECIMAL] string
LOW [FILL] [HEXADECIMAL] string

Specifies the high and/or low fill character used to formulate SQL queries for generic (@ and @@) search criteria.

If you use HEXADECIMAL, the string must be two hexadecimal digits. If you don't use HEXADECIMAL, the string must be a single ASCII character.

The defaults are database specific:

Database	HIGH fill character	LOW fill character
ALLBASE/SQL	HEX "FF"	HEX "00"
DB2	HEX "5A"	HEX "00"
MS SQL Server	HEX "7A"	HEX "00"
ODBC	HEX "FF"	HEX "00"
Oracle Rdb	HEX "FF"	HEX "00"
ORACLE	HEX "7F"	HEX "01"
SYBASE	the highest printable character on the machine	HEX "20"

Limit: Valid only for ORGANIZATION RELATIONAL (databases).

For information about using the low and high fill options, see (p. 68).

KEYFILE filespec (MPE/iX)

Indicates the name to be used by QUTIL when creating a KSAM key file. If the option is omitted, QUTIL creates a default from the file open name.

Using the FILE statement's KEYFILE option with KSAMXL files causes an error.

Limit: Valid only for KSAM files.

NULL VALUES [NOT] ALLOWED

Determines whether PowerHouse enables null value support for a particular database. If support is enabled, PowerHouse maintains null values in expressions, conditions, and aggregate functions, and nulls are stored when items are not initialized. If support is disabled, null character items are initialized to spaces (or dictionary initial values), and null date or numeric items are initialized to zeros (or dictionary initial values).

Default: NULL VALUES NOT ALLOWED

OPEN filespec|open-name-string

Specifies a valid filespec or database that is accessed and associated with the name that identifies it to the PowerHouse dictionary.

filespec

Specifies a valid file specification. It can be a physical file or database name, a file equation (**MPE/iX**), a logical name (**OpenVMS**), or an environment variable (**UNIX, Windows**).

MPE/iX:	For ALLBASE/SQL, the filespec must point to the root DBEnvironment. The PDL file name is truncated at the first non-alphanumeric character or after the first eight characters (six with IMAGE), whichever occurs first.
OpenVMS:	Files of type MBX (mailbox) can use only logical names, not file specification. The square brackets are required when you enter a directory name. Limit: Can contain up to 70 characters. Can use any valid combination of node, device, directory, extension, and version number. Consult OpenVMS documentation for more information about different parts of the file specification.
UNIX, Windows:	An environment variable must be preceded by a dollar sign ($). Limit: This form of the OPEN option cannot be used to identify databases other than Eloquence.

Default: The default filespec is the name after the FILE keyword.

filespec (Eloquence)

The format of an OPEN name for Eloquence is

[[*server_name*][:*service_name*]/]*database_name*

For example, nt001:eloqdb/qdb

Server_name is the name or IP address of the system running the database server. If the server is not specified in the PowerHouse open name, Eloquence first checks the value of the environment variable, EQ_DBSERVER. If EQ_DBSERVER is not set, the value of server defaults to the local system.

Service_name is the service name or port number of the database server. If it is not included in the open name, the Eloquence software uses its own rules to determine which service to use. It first checks the value of the environment variable, EQ_DBSERVER and, if that's not set, the default service name "eloqdb" is assumed.

open-name-string (OpenVMS, UNIX, Windows)

A string which is passed directly to the database server in order to gain access to the database.

An open-name-string contains delimited parameters such as userid, password, physical database name, network connection parameters, and possibly other parameters. If you include a userid and password in the open-name-string, it must resemble the following format:

SYBASE	<*database_physical_name*>@userid/password
ORACLE	ORACLE@userid[@<*network _connection_parameters*>]/password
DB2	not allowed
ODBC	not allowed

Please refer to your database documentation for more details on acceptable parameters, format, and syntax for your particular database's valid open specifications.

UNIX, Windows: The open-name-string may be an environment variable, which must be preceded by a dollar sign ($). If the string contains a required dollar sign, which is not used to specify an environment variable, use the backslash (\) to interpret it literally. For example:

```
ORACLE@OPS\$<userid>
```

For ORACLE databases, the string, "ORACLE@", is inserted at the beginning of the supplied open-name-string, if it does not exist. The at-sign is a delimiter used if other options are specified.

Default: If no OPEN options are specified, the database server will look for default environment variables or logicals that are specific to running that database's environment.

OWNER name

Specifies the owner for tables in a database when none is explicitly indicated. Also specifies the default owner of modules created by PowerHouse in ALLBASE/SQL.

Some relational databases support owners for entities such as modules or tables. If a program needs to access an entity owned by another user, you specify the owner as part of the entity name.

Limit: Valid only for ORGANIZATION RELATIONAL (databases).

Defaults: username (**MPE/iX**) or logonid (**OpenVMS, UNIX, Windows**).

Note: The program parameters, **noshift**, **upshift**, **downshift**, and SET statement shifting options that are in use may affect the case-sensitivity of the name literal.

PASSWORD string (MPE/iX)

Specifies the password required by PowerHouse for read/write access to the entire database.

Limit: 8 characters. Valid only for IMAGE.

Default: If the password is not supplied, PowerHouse uses a semicolon (;) for accessing or building a database, but only the database creator has write access, regardless of the file and element security specified. All others have read only access.

PASSWORD string (OpenVMS, UNIX, Windows for ORGANIZATION RELATIONAL)

If the password is not included in the open-name-string of the OPEN option, the PASSWORD option specifies the password to be used to connect to the database server. Passwords are set up by the database administrator.

PowerHouse combines the open-name-string, USERID, and PASSWORD options into a valid database open specification. The separator before a password is a slash (/), which PowerHouse inserts if the password does not start with it.

UNIX, Windows: The string may be an environment variable, which must be preceded by a dollar sign ($).

This option is required if a password is not included in the OPEN open-name-string option or in an associated DATABASE resource file statement.

PASSWORD string (UNIX, Windows for ORGANIZATION DATABASE)

For Eloquence, this clause is used to specify a master password for the database. For PowerHouse to work correctly, this password must give full read/write access to all data in the database. It must be a string of the form

[*password*]/*user*

This must match an already existing "user" within the Eloquence DBEnvironment. This is true whether the database is created outside of PowerHouse or is created by QUTIL.

If the Eloquence database is to be created by QUTIL, the specified "[password]/user" must identify a user with DBA (Database Administrator) privileges in the DBEnvironment. QUTIL generates the correct code to grant this user full access to all data in the database.

If the Eloquence database was created by an outside process, the "[password]/user" only needs to identify a user with full access to the data and DBA privilege is not required.

If no master password is provided for an Eloquence database, the Eloquence utilities use their own rules to determine what user(s) to use. They first look for the environment variables EQ_DBUSER and EQ_DBPASSWORD. If these are not found, the utility defaults are used. All of the utilities used by QUTIL will use the default "dba" user. Data access from the other products will use the default "public" user.

Note that, if no master password is provided for an Eloquence database in PDL, QUTIL may still be able to create the database, but will not be able to grant access to the data to the database creator.

RECORD FORMAT FIXED|VARIABLE (OpenVMS)

Establishes whether the file contains fixed-length or variable-length records. Direct files must have a fixed format.

Limit: Not valid for relational files.

Default: The default for MBX files is variable; the default for other files is fixed.

REUSE (MPE/iX)

Enables the record reuse capability of Native Mode KSAM files. This option causes an error if used with KSAM file types.

Limit: Valid only with KSAMXL file types.

Default: No reuse capability.

SCOPE TEMPORARY|PERMANENT (OpenVMS)

Determines how long a file of type MBX (mailbox) exists after it is created. You must have the OpenVMS system privilege PRMMBX and SYSNAM in order to create a permanent mailbox file in PowerHouse.

Temporary mailboxes are deleted by OpenVMS automatically when no channels are assigned to it; that is, when no one is using it. Use QUTIL to delete permanent mailboxes.

For more information about mailboxes, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Limit: Applies only to mailbox files.

Default: TEMPORARY

SUPERCEDE (OpenVMS)

Specifies that existing versions of the file should be overwritten each time the file is created.

SUPERCEDE is used in conjunction with the version number on the open name to designate whether the creation of a file will or will not overwrite an existing version of the file.

	Supersede=Yes	**Supersede=No**
Open name includes a version number	The existing data file is overwritten	An error message is issued
Open name does not include the version number	A new version of the data file is created	A new version of the data file is created

Default: By default, the PhD Screen System doesn't overwrite existing versions.

USERID identifier|string (OpenVMS, UNIX, Windows)

If the userid is not included in the open-name-string of the OPEN option, the USERID option specifies the userid to be used to connect to the database server. Userids are set up by the database administrator.

PowerHouse combines the open-name-string, USERID, and PASSWORD options into a valid database open specification. The separator before a userid is an at-sign (@), which PowerHouse inserts if the userid does not start with it.

UNIX, Windows: The string may be an environment variable, which must be preceded by a dollar sign ($).

Limit: Used only for SYBASE and ORACLE databases. This option is required if a userid is not included in the OPEN open-name-string option or in an associated DATABASE resource file statement.

Discussion

The FILE statement is used to declare a PowerHouse file. PowerHouse uses the file definition when opening, closing, creating, and deleting files. Every file must have a name and an organization. Files are referenced in PowerHouse by the name that you assign. The actual physical and logical files that PowerHouse gets its data from are defined by the file open name.

The FILE statement can also be used to identify a relational database to the PowerHouse dictionary. However, the DATABASE statement is recommended.

For more information on relational databases, see the *PowerHouse and Relational Databases* book.

Relationship Between ORGANIZATION and TYPE

File organization and file type are related. For example, relational databases must use the organization, RELATIONAL. Mailbox files (type MBX) must use the organization, SEQUENTIAL. RMS files can use any organization except RELATIONAL. MPE/iX and OpenVMS files can use RELATIVE organization, whereas, UNIX and Windows files cannot.

File organization is required in the FILE statement, but file type is required only for RELATIONAL. If the file type is not specified, PowerHouse determines it from the file organization. When the file type is specified, PDL confirms the appropriate type for the file organization.

The following table illustrates the relationship between file organization and file type:

Organization	Default Type	Alternate Type
RELATIONAL	no default	**MPE/iX:** ALLBASE **OpenVMS:** ORACLE, RDB, RDB/VMS **UNIX:** ORACLE, SYBASE, ODBC, DB2 **Windows:** ORACLE, DB2, ODBC, SYBASE
DATABASE	IMAGE (on MPE/iX); ELOQUENCE (on UNIX, Windows)	
DIRECT, SEQUENTIAL	MPE/UNIXIO/RMS/DOS	
RELATIVE	MPE/RMS	
INDEXED	KSAM/C-ISAM/RMS/DISAM	KSAMXL, KSAM64

Compatibility Between Eloquence and IMAGE

Although PowerHouse treats Eloquence databases as if they were IMAGE databases, there are some functional differences based on the internal structures of the two databases. Most of these are hidden because Eloquence traps the IMAGE intrinsic calls and maps them to Eloquence calls. For details of the internal differences, see the following page on the Marxmeier Web site:

http://www.hp-eloquence.com/hp3k

Note that an Eloquence database will not have the same record numbers as the corresponding IMAGE database nor does Eloquence use the same algorithms for positioning MASTER dataset entries. Synonym chains and migrating secondaries do not exist in Eloquence. Applications should not depend on record numbers or the ability to return records in a specific sequence before and after adds, updates, and deletions.

IMAGE Emulators (UNIX, Windows)

PowerHouse 4GL supports Eloquence. Other third party IMAGE emulators exist that use the same call interface. This call interface is available on all supported UNIX platforms (HP-UX, AIX, and Solaris) as well as Windows. As a result, PowerHouse treats other third party IMAGE emulators as it treats Eloquence, as an IMAGE database. Cognos has not tested third party IMAGE emulators and does not directly support them. If an issue arises that can be duplicated using Eloquence, it will be treated as a PowerHouse 4GL or PowerHouse Web issue. If it cannot be duplicated using Eloquence, it will be treated as an issue with the third party IMAGE emulator. Cognos will work with the third party to resolve the issue.

LOW and HIGH FILL Options

When a user specifies generic search criteria, PowerHouse uses fill characters in the BETWEEN clauses of queries.

```
>   SQL DECLARE EMPLIST CURSOR FOR &
>   SELECT EMPLOYEE, FIRST_NAME, LAST_NAME, &
>      BRANCHES.BRANCH, BRANCH_NAME &
>      FROM EMPLOYEES, BRANCHES &
>      WHERE EMPLOYEES.BRANCH = BRANCHES.BRANCH
>   ACCESS EMPLIST
>   CHOOSE FIRST_NAME PARM PROMPT "Enter a name: "
.
.
.
```

If the user enters K@ as the generic search criteria in the preceding example, the following BETWEEN clause of the SQL query is built using low and high fill characters:

```
BETWEEN 'Kll' AND 'Khh'
```

where ll is the low fill character and hh is the high fill character. The default fill characters depend on the type of database.

If the search criteria is K@@, the BETWEEN clause is

```
BETWEEN 'Kll' AND 'hh'
```

For more information, see "DB2: Determining the High and Low Fill Characters" (p. 37).

INDEX

Declares an index in the last-declared record-structure.

Syntax

INDEX name [option]...

name

Specifies a unique name for an index.

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters that are defined in the dictionary.

Options

INDEX options		
`ALTERNATE|PRIMARY`	`ASCENDING|DESCENDING`	`LINKS TO`
`NULL`	`OMNIDEX|TPI`	`ORDERED|UNORDERED`
`REPEATING|UNIQUE`		

ALTERNATE|PRIMARY

Specifies whether the index is an alternate or primary index. An indexed record-structure always contains one primary index, and can also contain alternate indexes. When an indexed file is defined, the first index becomes the primary index, by default, and all subsequent indexes become alternate indexes:

MPE/iX:	This option does not apply to IMAGE master and automatic master datasets.
OpenVMS:	For RMS files, primary indexes cannot be changed. PowerHouse deletes and adds any record whenever the index value for that record needs to be updated.

ASCENDING|DESCENDING

Indicates the order in which segments are stored in the index.

MPE/iX: ASCENDING can be specified, but is for documentation only. DESCENDING is not valid on MPE/iX.

UNIX, Windows: The default is overridden when you specify the ASCENDING or DESCENDING option on the SEGMENT statement. For information on using the ASCENDING and DESCENDING options of the SEGMENT statement, see (p. 95).

Limit: The ASCENDING option results in an error for IMAGE (**MPE/iX**).

Default: ASCENDING (**OpenVMS**).

LINKS TO record-structure [SORT ON item] (MPE/iX, UNIX, Windows)

Required for, and only valid with, IMAGE and Eloquence detail datasets. LINKS TO identifies which master or automatic master the detail dataset is associated with. SORT ON defines the IMAGE/Eloquence sort-item for the index that must be defined in the record-structure for the index, but cannot be a segment within this index, nor can a sort-item be part of the multi-segmented index within a record.

NULL [VALUE [IS] char] (OpenVMS)

Specifies that records with a null value for the index item are not to be indexed by that key. The null value for numeric and date items is always 0. By default, the null value for character items is 0, but this can be changed with the VALUE option. Not indexing null values can often save unnecessary overhead.

When the NULL option is not specified, all records are indexed, regardless of the index value. If the NULL option is specified, no index entry is created for records whose index value is null. Null is valid only for alternate indexes.

OMNIDEX|TPI (MPE/iX, UNIX, Windows)

Indicates a TPI (Third Party Indexing) or OMNIDEX sorted-sequential index within PowerHouse. PowerHouse supports TPI and OMNIDEX index retrievals directly through IMAGE or Eloquence. TPI and OMNIDEX provide the same functionality.

Because TPI indexes can only be associated to IMAGE or Eloquence databases, you must declare your files with an organization of DATABASE.

Limit: Not valid with the following options: ASCENDING, LINKS TO, ORDERED, and UNIQUE.

ORDERED|UNORDERED (MPE/iX)

Specifies whether the index records for duplicate keys are stored in chronological or random order.

Limit: Valid only for KSAM or KSAMXL indexes.

REPEATING|UNIQUE

REPEATING specifies that records may have the same index values. UNIQUE specifies that every record in the file must have a unique index value.

MPE/iX: Indexes in IMAGE master and automatic master datasets are always unique; the REPEATING option does not apply. The UNIQUE option is incompatible with KSAM or KSAMXL indexes if the ORDERED or UNORDERED options have been specified.

Default (**MPE/iX**): REPEATING for all files except IMAGE master and automatic master datasets.

Discussion

The INDEX statement defines an index, which is made up of between one and eight segments. A change to any of the index attributes requires you to unload, create, and reload any associated files and record-structures.

The INDEX statement is not valid with DIRECT, RELATIVE, SEQUENTIAL, or RELATIONAL file organizations.

MPE/iX, UNIX, Windows: With IMAGE and Eloquence, each master and automatic master must have exactly one index but a detail dataset can have 0-16 indexes. The index for an automatic master dataset is the entire record.

MPE/iX: With KSAM or KSAMXL, at least one index must be declared, and a maximum of 16 physical indexes can be declared. (Although PDL does not enforce this restriction, QUTIL passes only 16 valid indexes.)

OMNIDEX Indexes

Segments in OMNIDEX indexes need not be contiguous or of the same data type.

OMNIDEX indexes in PowerHouse support the following retrieval mechanisms:

- fixed-value range retrieval
- relational retrieval (<,<=,>,>=) for single-segment ASCII indexes
- PowerHouse generic (partial-key) retrieval for single-segment ASCII indexes
- OMNIDEX wildcard-character retrieval (#, ?, @) for single-segment ASCII indexes

Before using OMNIDEX indexes for PowerHouse retrievals, you must create them with OmniUtil. For more information about OmniUtil, see the DISC documentation.

The OMNIDEX index definition in PDL must match the OmniUtil definition exactly. PDL does not report discrepancies between the two definitions at parse time.

OMNIDEX indexes are verified at run time.

Index Compatibility in Eloquence and IMAGE

On MPE/iX for IMAGE, PowerHouse provides support for both B-Tree indexes and Third Party Indexing (TPI) indexes. B-Tree indexes are part of Hewlett Packard's standard IMAGE product and are identified in PDL using the INDEXED option on the RECORD statement for a MANUAL or AUTOMATIC MASTER dataset. QUTIL creates B-Tree indexes based on the INDEXED option. TPI indexes are extensions to the database provided by third party products. These are identified in PDL using the TPI or OMNIDEX option on the INDEX statement, but are not created by QUTIL. They must be added to the database after it is created using a third party utility. The functionality provided by B-Tree and TPI indexes is similar, but not identical.

In Eloquence, most TPI functionality is provided by the database without any need for a third party product. As is the case with IMAGE, TPI indexes are not created by QUTIL, but must be added after the database is created using the Eloquence utility, dbutil. At the database level, all indexes are implemented using the same technology. The low level calls that PowerHouse makes for B-Tree index access can be used with both the MASTER dataset search items created by QUTIL and the extra indexes added using dbutil to give results that are compatible with B-Tree access in IMAGE. Similarly, the low level calls that PowerHouse makes for TPI access can be used with both types of index to give results that are compatible with TPI access in IMAGE.

It is not possible for an index to be both TPI and B-Tree, either in IMAGE or in Eloquence.

PowerHouse support for Eloquence mimics IMAGE as much as possible even where Eloquence does not have the same restrictions as IMAGE. This was done so that the functionality and user expectations of an application migrated from IMAGE to Eloquence matches the original as much as possible. In order to provide this compatibility, the following rules apply to Eloquence.

For B-Tree indexes,

- You must specify INDEXED on the RECORD statement for your MASTER file.
- QUTIL creates B-Trees.
- Generic or ranged retrieval in QUICK will stop retrieving records as soon as a record is changed. Although this restriction is not required for Eloquence, it matches the IMAGE restriction. This does not apply to TPI.

For TPI indexes,

- You must specify TPI or OMNIDEX on the INDEX statement.
- QUTIL does not create TPI indexes. They must be added using a product specific utility.

How PowerHouse Creates Indexes

As it is more efficient to maintain fewer indexes, PowerHouse will merge indexes where possible.

If multiple indexes for a record-structure share the same initial subset of segments, then none of the shorter indexes can be the PRIMARY index.

PowerHouse will create one physical index for the longest index if two or more logical indexes meet all of the following criteria:

- the segments of the smaller indexes are a subset of the segments in the longer index
- the segments are in the same order
- the indexes have the same initial segment
- their attributes do not conflict (such as index ordering) and do not impact index integrity. (**OpenVMS, UNIX, Windows**)

OpenVMS, UNIX and Windows Example

Consider index X that is made up of segment A, index Y that is made up of segments A and B, and index Z that is made up of segments B and A.

A physical index will be created for index Z since the initial segment is different from X and Y. Whether one or two physical indexes for indexes X and Y is created is dependent on the index attributes for X and Y. If index X is repeating, then PowerHouse will only create one physical index for index Y since it can be used for both indexes. If, however, index X is unique, two physical indexes are created since a separate index is needed for index X to ensure the entries for segment A are unique.

MPE/iX

For KSAM and IMAGE

- the segments in the index must be contiguous. For example, if you have a Record X with Item A, Item B and Item C, then Index ABC is valid, but Index AC is not.
- the segments in the index are in the same order as the Record. For example, if you have Record X with Item A, Item B, and Item C, then Index BA would not be valid because the segments in the index don't appear in the same order as the corresponding items in the Record.

Limits:

MPE/iX:	For IMAGE, indexes cannot overlap. OMNIDEX indexes are not created by PowerHouse. They must be added to the database using OmniUtil.
OpenVMS:	Even though "n" number of logical indexes may be described, there is a limitation that only 8 physical indexes may be defined against an RMS file.
UNIX, Windows:	There is no limit to the number of indexes that a C-ISAM or DISAM file can have, however, it is more efficient to maintain fewer indexes. The number of segments a C-ISAM or DISAM file can have is 8, with the total segment size a maximum of 120 bytes.

Examples

In this generic example, ORDER-DETAIL is a repeating index; the same account number can occur in more than one record.

```
> FILE ORDER-DETAIL &
>      ORGANIZATION INDEXED &
>      OPEN ORDERDET
>
> RECORD ORDER-DETAIL
>      ITEM ORDER-NUMBER
>      ITEM CUSTOMERKEY
>      ITEM PART-NUMBER
>      ITEM PART-VARIANT
>      ITEM QUANTITY-ORDERED
>      ITEM QUANTITY-SHIPPED
>
>      INDEX ORDER-DETAIL &
>         PRIMARY REPEATING
>      SEGMENT ORDER-NUMBER
>
>      INDEX ORDER-PART &
>         ALTERNATE REPEATING
>      SEGMENT PART-NUMBER
>      SEGMENT PART-VARIANT
```

To define an index on two items (a multi-segment index):

```
> FILE POSITION &
>      ORGANIZATION INDEXED
>      DESCRIPTION "This file contains" &
>      "a record of each position. The POSITION" &
>      "is uniquely identified by DIVISION." &
>      "For example, the position programmer (PRG)" &
>      "can exist many times. However, the entry PRG" &
>      "marketing division (MKT) is unique."
> RECORD POSITION
>      ITEM POSITION
>      ITEM DIVISION
>      ITEM POSITION-TITLE
>
>      INDEX POSITION UNIQUE
>      SEGMENT POSITION
>      SEGMENT DIVISION
```

In the preceding example, POSITION is a unique index; only unique positions can be entered. POSITION is the most significant segment in the index.

MPE/iX

The following declares three OMNIDEX indexes on the BILLINGS IMAGE database. In this example:

- EMPLOYEE-NO indicates a non-OMNIDEX index that represents an IMAGE search item.
- EMPNO-OMNIDEX indicates an OMNIDEX index, for which the segment is EMPLOYEE-NO.
- PROJNO-OMNIDEX indicates a multi-segment OMNIDEX index. Up to eight segments can be declared in an OMNIDEX index.
- CPY-OMNIDEX indicates a single-segment ASCII OMNIDEX index.

```
> FILE BILLINGS ORGANIZATION DATABASE TYPE IMAGE
>
> RECORD BILLINGSINDEX ORGANIZATION AUTOMATIC
>   ITEM EMPLOYEE-NO
>
>   INDEX EMPLOYEE-NO UNIQUE
>   SEGMENT EMPLOYEE-NO
>
> RECORD BILLINGS ORGANIZATION DETAIL
>   ITEM EMPLOYEE-NO
>   ITEM MONTH
>   ITEM PROJECT-NO
>   ITEM BILLING
>   ITEM COMPANY
>
>   INDEX EMPLOYEE-NO LINKS TO BILLINGSINDEX
>   SEGMENT EMPLOYEE-NO
>
>
>
>
>
>   INDEX EMPNO-OMNIDEX OMNIDEX
>   SEGMENT EMPLOYEE-NO
>   INDEX PROJNO-OMNIDEX OMNIDEX
>   SEGMENT EMPLOYEE-NO
>   SEGMENT PROJECT-NO
>   SEGMENT BILLING
>
>
>
>   INDEX CPY-OMNIDEX OMNIDEX
>   SEGMENT COMPANY
```

ITEM

Declares an item in the last declared record-structure.

The ITEM statement is not valid for relational files.

Syntax

ITEM element [option]...

element

Names an element to be included as an item in the record-structure. The element must have already been declared in an ELEMENT statement.

Options

The options are CREATE, DATATYPE, OCCURS, REDEFINE, and SELECT.

CREATE

Instructs QUTIL to create this item when creating a file. Useful for specifying which item to create in an item redefinition.

DATATYPE type [SIGNED|UNSIGNED] [SIZE n [BYTES]]

Establishes how item values are stored. The type and size of the corresponding element determines the allowed and default item datatypes and sizes.

DATATYPE type

The PowerHouse datatypes are:

CHARACTER	DATETIME	FLOAT
FREEFORM	G_FLOAT (**OpenVMS**)	INTEGER
INTERVAL	JDATE	PACKED
PHDATE	VARCHAR	VMSDATE (**OpenVMS**)
ZDATE	ZONED	

Limit: Items based on character elements must use the datatype CHARACTER or VARCHAR, which can accommodate any ASCII letter, digit, symbol, or space. (There is no restriction, however, against using non-ASCII characters in CHARACTER or VARCHAR fields.)

Items based on date elements can use the datatype DATETIME, JDATE, PHDATE, VMSDATE (**OpenVMS**), ZDATE, INTERVAL, or any numeric datatype. Items based on numeric elements can use FLOAT, FREEFORM, INTEGER, PACKED, ZONED, or INTERVAL.

Default: PowerHouse sets the datatype according to the following table:

Element type	Element size	Default datatype
CHARACTER	n/a	CHARACTER
NUMERIC	1 to 9	INTEGER
NUMERIC	10 and over	FLOAT
DATE	6	PHDATE
DATE	8	INTEGER

NUMERIC

Indicates that the datatype, ZONED, is to have a type of LEADING SIGN NUMERIC rather than RIGHT OVERPUNCHED NUMERIC.

Limit: Valid only for ZONED datatypes.

SIGNED|UNSIGNED

Indicates whether the INTEGER, PACKED, and ZONED datatypes are SIGNED or UNSIGNED. INTEGER SIGNED can store negative values, but INTEGER UNSIGNED cannot. ZONED and PACKED datatypes can store positive or negative numbers, regardless of whether SIGNED or UNSIGNED is specified. The SIGNED and UNSIGNED options are mutually exclusive and must immediately follow the datatype.

Limit: Valid only for INTEGER, PACKED, and ZONED. SIGNED and UNSIGNED must immediately follow the datatype.

Default: SIGNED for datatypes INTEGER and PACKED; UNSIGNED for ZONED.

SIZE n [BYTES]

Specifies the storage size of the item.

Default: The default storage size depends on the element size.

The following table illustrates element size and default item size for character and numeric datatypes:

Datatype	Element Size	Default Item Size
CHARACTER	1 to 2047	same
FLOAT	1 to 6	4
	7 to 16	8
FREEFORM	1 to 31	same
INTEGER	1 to 4	2
	5 to 9	4
	10 to 14[1]	6
	15 to 31[1]	8
	10 to 31[2]	8
INTERVAL	16	8
PACKED	1 to 31	FLOOR(n/2) + 1
ZONED	1 to 31	same

[1] *PDL and PHDPDL with INTSIZE6 (**OpenVMS**)*

[2] *PHDPDL and PDL with NOINTSIZE6 (**OpenVMS**)*

The following table illustrates element size and default item size for date datatypes:

Datatype	Element Size	Default Item Size
DATETIME	16	8
FLOAT	6[1]	4
	8[1]	8

Datatype	Element Size	Default Item Size
INTEGER	6 or 8	4
JDATE	6 or 8	2
PACKED	6	4
	8	5
PHDATE	6 or 8	2
VMSDATE (**OpenVMS**)	8	8
ZDATE	8	6
ZONED	6	6
	8	8

[1] *Dates with four-digit years have element size 8; dates with two-digit years have element size 6.*

For more information about item datatypes and sizes, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

OCCURS n

Specifies the number of times an item repeats (in an array).

Limit: 1 to 4096. QDESIGN has a limit of 255 occurrences.

Default: 1

REDEFINES [ITEM] item

Indicates that this item redefines the most recently declared item on the same level.

Limit: ITEM is for documentation only.

SELECT value

Assigns a selection value that identifies a particular record-structure in a file that contains more than one record-structure. You must specify a value that's consistent with the type and size of the element. You can declare a selection value for more than one item in a record-structure.

Items that are defined with SELECT values serve as identifiers for coded record-structures. When the record-structure is accessed by PowerHouse, each record in the file is read, but only those records with the appropriate selection value are retrieved. Selection values also serve as unalterable initial values when a record is created.

Limit: Eight characters for CHARACTER items; nine digits for NUMERIC items.

Discussion

The ITEM statement defines an item in a record-structure. All of the ITEM statements for a particular record-structure must come immediately after the RECORD statement. PDL does not define elements automatically from item definitions, so the elements that are used in ITEM statements must already have been defined in ELEMENT statements. Note that the maximum number of items allowed in a record-structure is 1023.

Examples

The following PDL statements set up a record-structure. In this example:

- The DATATYPE option is specified on the numeric items within the substructure. Otherwise, the items would not contain the correct portion of the substructured data item.

- In this example, using the REDEFINES option allows the employee number to be referenced as either numeric or character, depending on the specified item name.

```
> FILE INVOICE-MASTER &
> ORGANIZATION INDEXED &
> OPEN INVOMAS
>
>ELEMENT DATE_INVOICE &
>    DATE CENTURY INCLUDED &
>    DEFAULT ITEM DATATYPE ZONED &
>    SIZE 8

> RECORD INVOICE-MASTER
>  ITEM INVOICE-NUM
>  ITEM CUSTOMER
>  ITEM EMPLOYEE
>  ITEM CUSTOMER-ORD-NUM
>  ITEM TERMS
>  ITEM INVOICE-PAID
>  ITEM DATE-INVOICE DATATYPE ZONED SIZE 8
>  BEGIN STRUCTURE
>  ITEM DATE-YEAR DATATYPE ZONED SIZE 4
>  ITEM DATE-MONTHDAY DATATYPE ZONED SIZE 4
>     BEGIN STRUCTURE
>     ITEM DATE-MONTH DATATYPE ZONED SIZE 2
>     ITEM DATE-DAY DATATYPE ZONED SIZE 2
>     END STRUCTURE
>  END STRUCTURE
.
.
> ELEMENT NEXT-EMPLOYEE NUMERIC SIZE 5
> ELEMENT NEXT-EMPLOYEE-CHAR CHARACTER SIZE 4
.
.
> FILE CONTROL ORGANIZATION DIRECT
>     RECORD CONTROL
>        ITEM NEXT-EMPLOYEE
>        ITEM NEXT-EMPLOYEE-CHAR &
>           REDEFINES NEXT-EMPLOYEE
```

The following example shows how to use the SELECT option of the ITEM statement to implement coded record-structures. SELECT causes this record-structure to only retrieve records from the file CONTACTS that have "C" in the RECORDTYPE item.

```
> FILE CONTACTS &
>     ORGANIZATION INDEXED &
>     OPEN CONTACTS
> RECORD CUSTOMERS
>     ITEM CUSTOMER-KEY
>        BEGIN STRUCTURE
>        ITEM RECORD-TYPE &
>           SELECT "C"
>        ITEM ACCOUNT-NUMBER
>        END STRUCTURE
>     ITEM CUSTOMER-NAME
>     ITEM STREET
>     ITEM CITY
>     ITEM PROVSTATE
>     ITEM POSTALZIP
>     ITEM PHONE-NUMBER
>
>     INDEX CUSTOMERS &
>        PRIMARY UNIQUE
>
>     SEGMENT CUSTOMER-KEY
```

LOAD

Loads the current PDL definitions into the dictionary.

Syntax

LOAD

Discussion

PDL and PHDPDL don't enter definitions into the dictionary when they first read and accept them. Instead, they store all the definitions in memory until the LOAD statement is entered and then load them into the dictionary all at once.

PDL and PHDPDL don't perform the load if they detect any errors up to the LOAD statement. After the LOAD statement, PDL and PHPDL perform an implicit CANCEL statement.

Examples

In the following example, the CREATE DICTIONARY statement creates a dictionary called INVENTOR. LOAD enters the definitions into the dictionary.

```
> CREATE DICTIONARY INVENTOR
>
> ELEMENT ACCOUNT-NUMBER NUMERIC SIZE 4
> ELEMENT LASTNAME CHARACTER SIZE 20
> ELEMENT FIRSTNAME CHARACTER SIZE 10
>
> FILE CUSTOMER ORGANIZATION INDEXED
> RECORD CUSTOMER
>        ITEM ACCOUNTNUMBER
>        ITEM LASTNAME
>        ITEM FIRSTNAME
>     INDEX ACCOUNTS UNIQUE
>     SEGMENT ACCOUNT-NUMBER
>
> LOAD
```

In a subsequent PDL or PHDPDL session, use the SET DICTIONARY statement to set the current dictionary to INVENTOR and add another file to the dictionary. Once you've done this, you can then enter the additional PDL statements that you require to define the new file and add them to the INVENTOR dictionary with the LOAD statement, as in

```
> SET DICTIONARY INVENTOR
>
> ELEMENT SALESREP CHARACTER SIZE 4
> ELEMENT LASTNAME CHARACTER SIZE 20
> ELEMENT REGION CHARACTER SIZE 10
>
> FILE SALES ORGANIZATION INDEXED
>
> RECORD SALES
>     ITEM SALESREP
>     ITEM LASTNAME
>     ITEM REGION
>     INDEX SALESREP UNIQUE
>     SEGMENT SALESREP
>
> LOAD
```

PERMIT

Allows or denies an application security class access to a record-structure or element.

Syntax

PERMIT asc TO capability [[,] capability]...
{ELEMENT|RECORD} name [[,] name]...

PERMIT capability [[,] capability]...
[OF] {ELEMENT|RECORD} name BY asc [[,] asc]...

asc

Names an application security class (ASC) that has been already defined in an APPLICATION SECURITY CLASS statement.

capability

Specifies the type of access permitted to the ASC for this record-structure or element. If you specify the ELEMENT option, the capabilities may be READ, CHANGE, or UNSPECIFIED. If you specify the RECORD option, the capabilities may be READ, WRITE, or UNSPECIFIED.

{ELEMENT|RECORD} name

Names a record-structure already defined in a RECORD statement, or an element already defined in an ELEMENT statement.

Discussion

The PERMIT statement allows specified ASCs access permission to record-structures or elements. A PERMIT statement with the RECORD option specified assigns record security; a PERMIT statement with the ELEMENT option specified assigns item security. To assign both record and item security, use more than one PERMIT statement.

The PERMIT statement cannot be used to define access rights to tables in a relational database. Granting permissions to elements will provide PowerHouse security on matching column names. Security settings for tables and columns are set in, and performed by the relational database to which they belong.

The PERMIT statement also denies access, since all ASCs not explicitly granted access are denied access.

If a user has write permission for a record-structure, he or she can read and write all items in the record-structure, and can create and delete records. Element security is ignored except for relational databases. Element security is applied to columns in a relational database regardless of a user's read or write access to the database.

If a user has no permission to access a record-structure and at least one application security class does have permission to access the record-structure, then the user is denied permission to access the record-structure.

If the user has read permission for a record-structure, then PowerHouse checks the user's item security for each item in the record-structure.

The following table summarizes how PowerHouse grants access permission:

Record Security	Other ASCs	Record Capability Granted	Element Security	Other ASCs	Element Capability Granted
Write	Ignored	Read, Update Insert, Delete	Ignored	Ignored	Read, Change

Record Security	Other ASCs	Record Capability Granted	Element Security	Other ASCs	Element Capability Granted
Read	Ignored	Read, Update	Read	Ignored	Read
			Change	Ignored	Read, Change
			Unspec	Yes	None
			Unspec	No	Read
Unspec	Yes	None	Ignored	Ignored	None
	No	Read, Update	Ignored	Ignored	Read, Change
		Insert, Delete			

All users belong, by default, to the application security class, UNKNOWN. The UNKNOWN class is a predefined class used to grant minimum capabilities to any user. You can assign additional permissions to this class the same as you would for any other class.

UNKNOWN is a preloaded ASC to which PowerHouse assigns any user that it cannot identify as a member of another ASC. If you give UNKNOWN explicit access to an element, you are effectively giving all ASCs at least the same access, because no ASC can have a lower level of access than UNKNOWN.

PERMIT doesn't override operating system security. If a user doesn't have operating system read or write permission for a file, he or she won't be able to access the file no matter what permissions you give with the PERMIT statement. This does not apply if you’re using Enhanced Application Security (**OpenVMS**).

PERMIT only affects users while they're using PowerHouse. A user who has operating system write permission to a file can use operating system utilities to change the file, even if you use PERMIT to deny them access to the file when they're using PowerHouse. Use PERMIT as a supplement to operating system security, not as a replacement.

Enhanced Application Security (OpenVMS)

Since PowerHouse application security specifications cannot override OpenVMS security, the security specifications in the dictionary do not always reflect the actual security encountered by users. To help improve this situation, PowerHouse has the ability to grant users access to any files to which the dictionary has access. This feature, called "Enhanced Application Security", is available only if PowerHouse has been installed with either the SYSPRV or GRPPRV privilege.

Under OpenVMS, each file carries a protection declaration that determines who has access to it. The protection declaration can recognize four classes of users:

Class of User	Description
OWNER	Is the user who created the file.
GROUP	Includes all users who belong to the same User Identification Code (UIC) group as the owner of the file. If the users are all members of the same UIC group, then GRPPRV may be used instead of SYSPRV.
WORLD	Includes all users.
SYSTEM	Includes all users and programs which • belong to the system group (by default, group numbers 1 to 10) • have the GRPPRV privilege and belong to the same group as the owner of the file they’re trying to access • have the SYSPRV privilege

Each class of user can be assigned one or more of the following capabilities: READ, WRITE, EXECUTE, and DELETE.

In most installations, the default file protection provides the SYSTEM with READ, EXECUTE, and WRITE capabilities so that backup and recovery can be performed. When PowerHouse is installed with SYSPRV, it is able to access file with the capabilities for SYSTEM.

Whenever an attempt is made to access a file using PowerHouse, PowerHouse checks the record-structure definition in the dictionary for security specifications. If there are any, and if the user satisfies them, PowerHouse opens the file with its SYSPRV privilege (which it immediately turns off again until next time) and checks the OpenVMS protection declarations for the file. PowerHouse allows the user access to the file if any of the following conditions are true:

- WORLD access is allowed.
- GROUP access is allowed and the UIC group number of the data file matches that of the dictionary owner or that of the user.
- OWNER access is allowed and the UIC group number of the data file matches that of the dictionary owner or that of the user.

In the last two cases, PowerHouse is extending its own privilege to access the file to the user because the dictionary owner has access to the file. Thus, through application security specifications, the dictionary owner can grant his or her own file access capabilities to any user.

Users could take advantage of this feature to grant themselves access to secured files by changing the security specifications in the dictionary. To prevent this, the dictionary owner (or any other dictionary manager) can use dictionary security. Simply deny such users access to application security functions within PhD Screen System or PHDPDL by assigning them to the USER dictionary class.

Enhanced Application Security is never applied to files whose record-structure definitions in the dictionary have no security specifications.

Examples

MPE/iX

In the following example, the users with login IDs "PRG" and "MANAGER" can read or write the INVENTORY record-structure. The final three PERMIT statements are evaluated together to determine access granted to the user CLERK. No other users can access the INVENTORY record-structure.

PowerHouse components use application security classes combined with PERMIT statements to determine access rights to record-structures and elements.

```
> SYSTEM OPTIONS &
>      ASC ID METHOD LOGONID
.
.
.
> ASC DEVELOPMENT &
>      LOGONID PRG.DOC
>
> ASC CLERK &
>      LOGONID CLERK.DOC
>
> ASC PRODUCTION &
>      LOGONID MANAGER.DOC &
>      LOGONID PRG.DOC
.
.
.
> PERMIT WRITE &
>      OF RECORD INVENTORY BY &
>      DEVELOPMENT &
>      PRODUCTION
.
.
.
```

```
> PERMIT READ &
>      OF RECORD INVENTORY &
>      BY CLERK
>      PRODUCTION
.
.
.
> PERMIT CLERK TO &
>      READ &
>      ELEMENT &
>         INVENTORY-NUM &
>         INVENTORY-ITEM &
>         UNIT-COST &
>         UNIT-MARKUP
>
> PERMIT CLERK TO &
>      CHANGE &
>      ELEMENT &
>      QTY-ON-HAND
```

UNIX, Windows

In the following example

- the users, "DEVELOPMENT" and "PRODUCTION", can read or write the INVENTORY record-structure.
- PowerHouse components use application security classes combined with PERMIT statements to determine access rights to record-structures and elements.
- The final three PERMIT statements are evaluated together to determine access granted to the user, "CLERK". No other users can access the INVENTORY record-structure.
- READ permission on a record-structure grants READ and UPDATE access to all items. However, when a READ permission on a record-structure is encountered, PowerHouse checks for item security.
- CLERK can read INVENTORYNUM, INVENTORYITEM, UNITCOST, and UNITMARKUP.
- CLERK can change the QTYONHAND field of the INVENTORY record-structure.

```
> SYSTEM OPTIONS &
>      ASC ID METHOD UIC
.
.
.
> ASC DEVELOPMENT &
>      UIC [26,123] [26,128]
>
> ASC CLERK &
>      UIC [26,185]
>
> ASC PRODUCTION &
>      UIC [26,190] &
>      [26,195]
.
.
.
> PERMIT WRITE &
>      OF RECORD INVENTORY BY &
>      DEVELOPMENT &
>      PRODUCTION
> PERMIT CLERK TO &
>      READ &
>      ELEMENT &
>         INVENTORYNUM &
>         INVENTORYITEM &
>         UNITCOST &
>         UNITMARKUP
>
> PERMIT CLERK TO &
>      CHANGE &
>      ELEMENT &
>      QTYONHAND
```

OpenVMS

In the following example

- the users with logon ID, "PRG", can read or change the INVENTORY record-structure.
- The logon ID, "MANAGER", can only read INVENTORY.
- The logon ID, "CLERK", can read the entire record-structure, but can only change the QTY_ON_HAND item. No other users can access the INVENTORY record-structure.
- PowerHouse components use application security classes combined with PERMIT statements to determine access rights to record-structures and elements.
- READ permission on a record-structure grants READ and UPDATE access to all items. However, when a READ permission on a record-structure is encountered, PowerHouse checks for element security for each item in the record-structure.
- The user PRG has full access to the listed elements since PRG is a member of both the PRODUCTION and DEVELOPMENT application security classes.
- As a member of the ASC PRODUCTION, the user MANAGER has only read access to the elements.
- CLERK can read INVENTORY_NUM, INVENTORY_ITEM, UNIT_COST, and UNIT_MARKUP.
- CLERK can change the QTY_ON_HAND field of the INVENTORY record-structure.

```
> SYSTEM OPTIONS &
>      ASC ID METHOD LOGONID
.
.
> ASC DEVELOPMENT &
>      LOGONID PRG
>
> ASC ENTRY &
>      LOGONID CLERK
>
> ASC PRODUCTION &
>      LOGONID MANAGER
>      LOGONID PRG
.
> PERMIT READ &
>      OF RECORD INVENTORY BY &
>      DEVELOPMENT &
>      PRODUCTION
>      ENTRY
>
> PERMIT DEVELOPMENT TO CHANGE ELEMENT &
>         INVENTORY_NUM &
>         INVENTORY_ITEM &
>         UNIT_COST &
>         UNIT_MARKUP &
>         QTY_ON_HAND
>
> PERMIT PRODUCTION TO READ ELEMENT &
>         INVENTORY_NUM &
>         INVENTORY_ITEM &
>         UNIT_COST &
>         UNIT_MARKUP &
>         QTY_ON_HAND
>
> PERMIT ENTRY TO READ ELEMENT &
>         INVENTORY_NUM &
>         INVENTORY_ITEM &
>         UNIT_COST &
>         UNIT_MARKUP &
>
> PERMIT ENTRY TO CHANGE ELEMENT &
```

```
                >      QTY_ON_HAND
```

QUIT

Ends a PDL or PHDPDL session.

Syntax

QUIT

Discussion

The QUIT statement ends the session and returns control to the operating system or to the invoking program.

QUIT and EXIT are interchangeable.

RECORD

Declares a record-structure.

Syntax

RECORD name [option]...

name

Specifies a unique record-structure name.

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters as defined in the SYSTEM OPTIONS statement.

Options

RECORD options		
CAPACITY	CREATE|NOCREATE	DESCRIPTION
INDEXED	OPEN	ORGANIZATION

CAPACITY n (MPE/iX, UNIX, Windows)

Declares the number of records QUTIL should allow for when creating the record.

Limit: Valid only with IMAGE or Eloquence datasets. With Eloquence, it is valid but ignored.

Default: 1,023 records

CREATE|NOCREATE

CREATE specifies that the record-structure is to be selected by QUTIL as the one which will be used to establish record length and index structures when the associated file is created. This is useful when more than one record-structure has been defined for a file. The rules are slightly different for records in DATABASE files than for records in non-DATABASE files. (**MPE/iX**)

NOCREATE can be used to designate record-structures that are not to be selected by QUTIL. QUTIL will then select among the other record-structures defined for the file.

If a file contains only one record-structure, QUTIL always declares that record-structure to the file creation utility, regardless of whether CREATE or NOCREATE has been specified for it.

If a non-DATABASE file contains more than one record-structure, QUTIL applies the following rules, in order, in deciding which one to declare:

1. If one record-structure has the CREATE option, and the other record-structures have the NOCREATE option or do not have any creation option, QUTIL uses the record-structure with the CREATE option.
2. If one of the record-structures has the same name as the file, QUTIL uses that one, regardless of its creation option.
3. If there is no other way to make the decision, QUTIL uses the record-structure that was entered into the dictionary first, regardless of its creation option.

For more information about the CREATE and NOCREATE options, see the Discussion on (p. 88).

DESCRIPTION string [[,] string]...

Specifies a description that serves as documentation about the record-structure. The description can be reported in QSHOW.

Limit: 60 characters per string.

INDEXED (MPE/iX, UNIX, Windows)

Specifies that the key of the master dataset is created with a B-Tree index.

Limit: This is valid only for IMAGE or Eloquence records with an organization of MASTER or AUTOMATIC MASTER.

OPEN name (MPE/iX, UNIX, Windows)

Declares the name of the IMAGE or Eloquence dataset for PowerHouse to use when accessing the record.

Limit: The name can be up to 16 characters. Valid only with IMAGE or Eloquence datasets.

Default: If not specified, a default name is created from the first 16 characters of the record name.

ORGANIZATION type (MPE/iX, UNIX, Windows)

Declares the type of IMAGE or Eloquence dataset. The options are MASTER, AUTOMATIC [MASTER], and DETAIL.

Limit: Valid only with IMAGE or Eloquence datasets.

Default: DETAIL

Discussion

The RECORD statement names a record-structure and lists its attributes. The RECORD statement must be entered immediately after the FILE statement that describes the file to which the record-structure belongs, or after another record-structure definition that belongs to the same file. The ITEM, INDEX, and SEGMENT statements for each record-structure must come immediately after their associated RECORD statements.

Coded Record-structures

To declare a file that contains more than one record-structure within the same physical file, first define the file using the FILE statement. Then declare each record-structure immediately after the FILE statement that describes the file to which the record-structure belongs, or after another record-structure definition that belongs to the same file. For example, when the DEDUCTIONS file contains the record-structures, MANDATORYDEDUCT and VOLUNTARY-DEDUCT, enter the two RECORD statements following a single FILE statement for the DEDUCTIONS file.

Selection values can be declared for each record-structure. PowerHouse uses the values to differentiate between the record-structures. Any item can be used as a record code, and any number of items can be used. To declare selection values for record codes, use the SELECT option of the ITEM statement.

Note (**MPE/iX**): The record-structures representing the datasets of an IMAGE database are not coded record-structures.

Combining Coded Record-structures

When combining different record-structures in the same file, you must ensure that the file management system can find the items for indexes which have been declared. This is only true if all such items are found in the part of the record-structure that is common to all of the different record-structures. When the variant part of a record-structure contains items on which indexes are declared, it is advisable to group these items at the start of the variant section.

Coded record-structures function best if you:

- place items common to all record-structures at the beginning of the record-structures
- place items that depend on codes at the end of the record-structures

You should avoid complex index structures whose segments correspond to items in the variant part of a coded record-structure. In such cases, you must ensure that the various coded record-structures overlap appropriately. If the items on which indexes are declared do not overlap properly, the file management system cannot handle the indexes correctly.

Filler data items are used to ensure that all of the records in a file containing coded record-structures are the same size. They are not necessary if you are using variable length records (**OpenVMS**). The record size consists of the total of all the item sizes in the record-structure.

For example, two record-structures, MANDATORY-DEDUCT and VOLUNTARY-DEDUCT, are combined in the DEDUCTIONS file. As each of the record-structures is defined, selection values are specified for the DEDUCTION-CODE item (which is included in both record-structures). The selection value for DEDUCTION-CODE in MANDATORY-DEDUCT is M. In VOLUNTARY-DEDUCT the selection value for DEDUCTION-CODE is V.

When the PowerHouse components access VOLUNTARY-DEDUCT, the DEDUCTIONS file is opened and each record is read. If a record has a value of V for DEDUCTION-CODE, the record is used. Records with a value of M are ignored.

A record-structure can have more than one item with a selection value. For example, record-structure XYZ could include the items A, B, C, and D. Item A has a selection value of 1; B has a selection value of 2. When record-structure XYZ is accessed, the only records that are used are those in which the item A has a value of 1 and B has a value of 2.

Coded records are said to have the same index layouts if the following are true:

- the number of indexes are the same
- the index sizes are the same
- the segment types are the same (note that multi-segmented indexes are treated as CHARACTER)
- the starting offsets are the same

The RECORD statement must declare a new record-structure; it can't reference an existing record-structure.

Discussion (MPE/iX)

The record layout of an IMAGE automatic master or master dataset must define exactly one index. Master and automatic master records must be declared before any detail records can reference them. Multiple RECORD statements with the same open name in the same IMAGE database are treated as coded records. Such records should have the same dataset organization, the same length in bytes, and the same index layout.

CREATE|NOCREATE for IMAGE DATABASE files

Multiple record-structures are used with DATABASE files to specify different datasets in an IMAGE database. An IMAGE database generally contains more than one dataset, so more than one record-structure is selected by QUTIL.

NOCREATE specifies that a record-structure can never be selected by QUTIL. If there is only one record-structure for a dataset (that is, only one RECORD statement with a given OPEN name), and NOCREATE is specified, the dataset can never be created by QUTIL.

If there is more than one record-structure for a dataset (that is, more than one RECORD statement with a given OPEN name) QUTIL applies the following rules, in order, in deciding which one to declare:

1. If one record-structure has the CREATE option, and the other record-structures have the NOCREATE option or do not have a creation option, QUTIL uses the record-structure with the CREATE option.
2. QUTIL uses the first record in alphabetical order.

Examples

This example is generic. For an MPE/iX example using an IMAGE database, see (p. 90).

The following example shows the creation of two record-structures, SALESREP and CUSTOMERS.

- The element SALESREP appears in both record-structures as an item.
- The first record-structure contains items that are in the file SALESREP.
- The second record-structure contains items that are in the file CUSTOMERS.

```
> ELEMENT SALESREP NUMERIC SIZE 4
> ELEMENT LASTNAME CHARACTER SIZE 20
> ELEMENT FIRSTNAME CHARACTER SIZE 10
> ELEMENT QUOTA NUMERIC SIZE 9 DECIMAL 2  &
>         PICTURE "^,^^^,^^^.^^"
> ELEMENT COMMISSION NUMERIC SIZE  4 DECIMAL 2 &
>         PICTURE "^^,^^%"
> ELEMENT CUSTOMER CHARACTER SIZE 5
> ELEMENT BILLING NUMERIC SIZE 7 DECIMAL 2 &
>         PICTURE "^^,^^^.^^"
> ELEMENT ADDRESS CHARACTER SIZE 60
> ELEMENT REGION NUMERIC SIZE 10
> ELEMENT AMOUNTOWING NUMERIC SIZE 7 DECIMAL 2
> ELEMENT AMOUNTPAID NUMERIC SIZE 7 DECIMAL 2
>
> FILE SALESREP ORGANIZATION INDEXED OPEN SALES
> RECORD SALESREP
>  ITEM SALESREP
>  ITEM LASTNAME
>  ITEM FIRSTNAME
>  ITEM QUOTA
>  ITEM COMMISSION
>  INDEX SALESREP UNIQUE
>    SEGMENT SALESREP
> FILE CUSTOMERS ORGANIZATION INDEXED OPEN CUST
> RECORD CUSTOMERS
>  ITEM CUSTOMER
>  ITEM ADDRESS
>  ITEM REGION
>  ITEM BILLING
>  ITEM SALESREP
>  ITEM AMOUNTOWING
>  ITEM AMOUNTPAID
>  INDEX CUSTOMERS UNIQUE
>    SEGMENT CUSTOMER
>  INDEX SALESREP REPEATING
>    SEGMENT SALESREP
> LOAD
```

To define coded record-structures in a single file:

```
> FILE CONTACTS &
>      ORGANIZATION INDEXED &
>      OPEN CONTACTS
>
> RECORD CONTACTS
>      ITEM CONTACTKEY
>         BEGIN STRUCTURE
>         ITEM RECORDTYPE
>         ITEM ACCOUNTNUMBER
>         END STRUCTURE
>      ITEM CONTACTNAME
>      ITEM STREET
>      ITEM CITY
>      ITEM PROVSTATE
>      ITEM POSTALZIP
>      ITEM PHONENUMBER
>      INDEX CONTACTS &
>         PRIMARY UNIQUE
>      SEGMENT CONTACTKEY
>      INDEX CONTACTNAME &
>         ALTERNATE REPEATING
>      SEGMENT CONTACTNAME
> RECORD CUSTOMERS
>      ITEM CUSTOMERKEY
>         BEGIN STRUCTURE
>         ITEM RECORDTYPE &
>             SELECT "C"
>         ITEM ACCOUNTNUMBER
>          END STRUCTURE
>      ITEM CUSTOMERNAME
>      ITEM STREET
>      ITEM CITY
>      ITEM PROVSTATE
>      ITEM POSTALZIP
>      ITEM PHONENUMBER
>      INDEX CUSTOMERS &
>          PRIMARY UNIQUE
>      SEGMENT CUSTOMERKEY
>
> INDEX CUSTOMERNAME &
>      ALTERNATE REPEATING
> SEGMENT CUSTOMERNAME
> RECORD SUPPLIERS
>      ITEM SUPPLIERKEY
>          BEGIN STRUCTURE
>          ITEM RECORDTYPE &
>             SELECT "S"
>      ITEM ACCOUNTNUMBER
>      END STRUCTURE
>      ITEM SUPPLIERNAME
>      ITEM STREET
>      ITEM CITY
>      ITEM PROVSTATE
>      ITEM POSTALZIP
>      ITEM PHONENUMBER
>      INDEX SUPPLIERS &
>          PRIMARY UNIQUE
>      SEGMENT SUPPLIERKEY
>      INDEX SUPPLIERNAME &
>          ALTERNATE REPEATING
>      SEGMENT SUPPLIERNAME
```

The values in the SELECT options determine which records are retrieved for each record-structure.

MPE/iX

For a generic example, see (p. 88).

An IMAGE database is defined using a single FILE statement followed by one RECORD statement for each dataset in the database. The IMAGE definition ends when PDL encounters another FILE statement.

```
> FILE INVDB ORGANIZATION DATABASE
>
> RECORD INVEN-ITEM ORGANIZATION MASTER &
>      OPEN INVDB01
>   ITEM STOCK-CODE
>   ITEM STOCK-DESC
>   ITEM PURCHASE-UNITS
>   ITEM BUYER-NO
>
>   INDEX INVEN-ITEM-IDX &
>      UNIQUE
>   SEGMENT STOCK-CODE
.
.
.
>   RECORD ITEM-LOC-AM ORGANIZATION AUTOMATIC MASTER &
>   OPEN INVDB03
>   ITEM STOCK-LOCATION
>
>   INDEX ITEM-LOC-AM-IDX &
>      UNIQUE
>      SEGMENT STOCK-LOCATION
.
.
```

```
.
> RECORD ITEM-DESC ORGANIZATION DETAIL &
>    OPEN INVDB07 &
>    DESCRIPTION " This file contains a variable " &
>    "number of lines of description for each " &
>    "inventory item. These descriptions will " &
>    "appear in the stock catalog"
> ITEM STOCK-CODE
> ITEM LONG-DESC
> ITEM SEQUENCE-NO
>
> INDEX ITEM-DESC &
>    LINKS TO INVEN-ITEM &
>    PRIMARY UNIQUE
>    SEGMENT STOCK-CODE
> FILE VENDOR ORGANIZATION INDEXED
> RECORD VENDOR
>    ITEM VENDOR-NUM
>    ITEM VENDOR-NAME
>    ITEM VENDOR-SHORT-NAME
>    ITEM VENDOR-ADDRESS
>    ITEM VENDOR-PHONE
>
> INDEX VENDOR-IDX &
>    UNIQUE
>    SEGMENT VENDOR-NUM
```

REVISE

Invokes the system editor to edit PDL source statement files.

Syntax

REVISE [*|filespec [option]...]

*

Specifies that PDL's temporary save file is to be revised.

You must use an asterisk when you want to revise PDL's temporary save file and also specify options. For example, you cannot revise the temporary save file with the NOUSE option by specifying

```
> REVISE NOUSE
```

because this tells PDL to look for a specific file named "NOUSE". To revise the source statement save file with the NOUSE option, enter

```
> REVISE * NOUSE
```

filespec

Names the file you want to revise.

Options

The options are DETAIL, NODETAIL, LIST, NOLIST, USE, and NOUSE.

DETAIL|NODETAIL

DETAIL writes the contents of the revised file to PDL's temporary file after the revision is completed; NODETAIL specifies that the contents are not written to the temporary file.

If a file other than PDL's temporary save file is being revised with USE and NODETAIL in effect, then a USE statement is written to PDL's temporary save file.

Limit: NODETAIL isn't valid with REVISE *.

Default: DETAIL

LIST|NOLIST

LIST displays the statements in the revised file as PDL processes them; NOLIST does not.

Default: LIST

USE|NOUSE

USE processes the revised statements when you exit from the system editor. NOUSE returns you to PDL at the point from which you left it, without processing the revised statements.

Default: USE

Discussion

The REVISE statement lets you use the system editor to edit either PDL's temporary save file or a permanent file from within PDL. The temporary save file is edited by default.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the REVISE statement. For more about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

When you enter REVISE without a filespec, PDL automatically performs a CANCEL CLEAR statement prior to processing the statements. If you enter REVISE with a filespec, the automatic CANCEL CLEAR statement isn't performed.

MPE/iX

By default, the REVISE statement uses the editor defined by the file equation, COGEDITR. If COGEDITR is not defined, the REVISE statement fails.

If you do not qualify the filename, PDL locates the file using a sequence of steps. For more information about locating files, see Chapter 1, "Running Powerhouse", in the *PowerHouse Rules* book.

OpenVMS

The REVISE statement invokes the DCL command assigned to the global symbol, PHEDIT (usually used to designate an editor). By default, the SET POWERHOUSE command sets PHEDIT to

```
$ PHEDIT :== EDIT/EDT
```

causing the REVISE statement to invoke the EDT editor.

You can change the default editor by changing the setting of the PHEDIT symbol. For example, to use the special interface to EDT called UTILITIES:EDT.COM, change the setting to

```
$ PHEDIT :== @UTILITIES:EDT.COM
```

We recommend that you use either EDIT/EDT or EDIT/TPU as the setting for PHEDIT. In either of these cases, the editor can be called directly; otherwise, a subprocess is spawned.

If you intend to use the **nodcl** program parameter to restrict user access to the operating system, we further recommend that you do not select editors (such as TPU) that provide operating system access. When **nodcl** is in effect, users will continue to be able to access the system editor through the REVISE statement.

For more information about setting global symbols and about the various editors that are available, see the OpenVMS General User's Manual.

UNIX, Windows

By default, the REVISE statement uses the editor defined by the environment variable, PHEDIT. If PHEDIT is not defined, the system checks the environment variable, EDITOR. If you have not defined either of these variables, the REVISE statement fails.

Example

To edit a permanent file, you must enter a filespec in the REVISE statement. The following statement

```
> REVISE PDLSRC
```

invokes the text editor and allows you to edit the file PDLSRC.

SAVE

Saves the current PDL statements in a file.

Syntax

SAVE filespec [CLEAR]

filespec

Names the permanent file that will be created to contain the PDL statements.

CLEAR

Removes any source statements in the temporary save file, once the contents are saved to a permanent file.

Discussion

The SAVE statement copies the contents of the temporary save file to the file that you name. PDL uses the temporary save file to record all PDL statements as they are entered (except the SAVE statement). The temporary save file is deleted when you exit from PDL.

The SAVE statement creates a new, permanent file and copies the contents of the temporary save file into it. You can then use the new file either as a source file for documentation and future changes, or as a working file for modification using the system editor. The saved statements can also be processed by PDL with the USE statement.

The CLEAR option clears the temporary save file after its contents have been saved so that you can enter and then save a new set of PDL statements in the same session. To clear the temporary save file without saving its contents, use SET SAVE CLEAR or CANCEL CLEAR.

SAVE saves everything. If you have entered statements with errors, those statements are included in the temporary save file. You should enter a CANCEL CLEAR statement after each error to clear the PDL temporary save file or use the REVISE statement to correct the errors. This ensures that the statements you save to a permanent file are error free. If you don't clear the file before you save statements, you might inadvertently include statements that you don't want.

Example

The SAVE statement saves all statements, including those with errors, to a permanent file called PDLSRC. The statements can then be modified using REVISE.

```
> CREATE DICTIONARY INVENTOR
>
> ELEMENT ACCOUNT-NUMBER NUMERIC SIZE 4
> ELEMENT LASTNAME CHARACTER SIZE 20
> ELEMENT FIRSTNAME CHARACTER SIZE 10
> FILE CUSTOMER ORGANIZATION INDEXED
> RECORD CUSTOMER
>         ITEM ACCOUNT-NUMBER
>         ITEM LASTNAME
>         ITTEM FIRSTNAME
          ^^^^^
*E* Expected: CANCEL CREATE EXIT QUIT REVISE SAVE SET SHOW USE <eol> ITEM

>      INDEX ACCOUNTS UNIQUE
>      SEGMENT ACCOUNT-NUMBER
>
> SAVE PDLSRC CLEAR
```

SEGMENT

Declares a segment in an index.

Syntax

MPE/iX, UNIX, Windows:	**SEGMENT item [ASCENDING|DESCENDING] [DATATYPE datatype [SIGNED|UNSIGNED] [SIZE n [BYTES]]]...**
OpenVMS:	**SEGMENT item [DATATYPE datatype [SIGNED|UNSIGNED] [SIZE n [BYTES]]]...**

item

Specifies the name of an item in the record-structure.

ASCENDING|DESCENDING (MPE/iX, UNIX, Windows)

Indicates the order in which segments are stored in the index.

Limit (**MPE/iX**)**:** The ASCENDING option is for documentation purposes only. The DESCENDING option is not valid.

If these options are not specified, the ASCENDING and DESCENDING options of the INDEX statement determine the order in which segments are stored. For information on the ASCENDING and DESCENDING options of the INDEX statement, see (p. 69).

DATATYPE datatype [SIGNED|UNSIGNED] [SIZE n [BYTES]]

Establishes how the segment value is stored. The datatype used must be equivalent to the datatype of the segment's corresponding item.

Limit: The DATATYPE option is for documentation only.

SIGNED|UNSIGNED

The option used must be equivalent to that of the segment’s corresponding item.

Limit: Applies only to INTEGER, PACKED, and ZONED. SIGNED and UNSIGNED must immediately follow the datatype.

Default: SIGNED for datatypes INTEGER and PACKED; UNSIGNED for ZONED.

SIZE n [BYTES]

Specifies the number of bytes to be reserved in each record for storage of the segment value. The size specified must be equivalent to the size of the segment's corresponding item.

Limit: The SIZE option is for documentation only.

Discussion

An index may have a maximum of eight segments defined.

OpenVMS: When two or more segments are used, the item datatype must be CHARACTER or ZONED.

The SEGMENT statements must come immediately after the INDEX statement that they belong to.

The segment statement is not valid with direct, relative (**MPE/iX, OpenVMS**) and sequential file organizations. A segment item cannot be an occurring item. That is, if an item is defined by the statement

```
> ITEM DEPT OCCURS 4
```

then the item DEPT cannot be used in a SEGMENT statement.

If the item DEPT in the example above had the OCCURS option on its corresponding ELEMENT statement, then the item DEPT cannot be used as a segment in an index.

Multi-Segment Indexes Limits (MPE/iX)

Segments of a multi-segment index must form a "contiguous block". The list of segments in the index must be an exact sequence of their corresponding items in the record-structure.

For KSAM and KSAMXL files, if multiple indexes for a record-structure share the same initial subset of segments, then none of the shorter indexes can be the PRIMARY index. IMAGE indexes cannot overlap.

Example

The following example shows how to define an index on two items (a multi-segment index). In this preceding example:

- POSITION is a unique index; only unique positions can be entered.
- POSITION is the most significant segment in the index.

```
> FILE POSITION &
>      ORGANIZATION INDEXED
>      DESCRIPTION "This file contains" &
>      "a record of each position. The POSITION" &
>      "is uniquely identified by DIVISION." &
>      "For example, the position programmer (PRG)" &
>      "can exist many times. However, the entry PRG" &
>      "marketing division (MKT) is unique."
>
> RECORD POSITION
>      ITEM POSITION
>      ITEM DIVISION
>      ITEM POSITIONTITLE
>
>      INDEX POSITION &
>         UNIQUE
>      SEGMENT POSITION
>      SEGMENT DIVISION
```

SET

Changes the default settings for a session.

Syntax

SET DEFAULT

SET option...

DEFAULT

In PDL, resets the SET statement options to the default values: DETAIL, LIST, NOPRINT, NOVERIFY ERRORS, VERIFY DELETE, and WARNINGS

In PHDPDL, resets the SET statement options to the default values: NOPRINT, NOVERIFY ERRORS, and WARNINGS.

If DEFAULT is used, no other SET statement options can be used.

Options

SET options

DETAIL|NODETAIL	DICTIONARY	DOWNSHIFT|UPSHIFT|NOSHIFT
LIST|NOLIST	PRINT|NOPRINT	SAVE CLEAR
VERIFY|NOVERIFY	WARNINGS|NOWARNINGS	

DETAIL|NODETAIL

The NODETAIL option prevents information from being loaded by way of a USE file into the temporary save file.

Limit: Valid in PDL; not valid in PHDPDL.

Default: DETAIL

DICTIONARY filespec

Establishes which PowerHouse dictionary to use for the current session.

OpenVMS: The dictionary logical(s) are set by this command to this dictionary.

Default: The PDL and PHDPDL compiler sets PHD as the default dictionary.

DOWNSHIFT|UPSHIFT|NOSHIFT

Specifies that the values of entered identifiers be shifted to lowercase, uppercase, or left as entered.

For system-wide access to mixed, lowercase, or uppercase identifiers, you can specify the SHIFT option in the SYSTEM OPTIONS statement.

If you specify NOSHIFT to PDL, you must also specify NOSHIFT in any other PowerHouse component where you require access to mixed or lowercase identifiers.

Limit: DOWNSHIFT, UPSHIFT and NOSHIFT are parsed in PHDPDL but you get a warning that UPSHIFT is being used. The **downshift**, **upshift**, and **noshift** program parameters are ignored in both PHDPDL and the PhD Screen System.

Default: UPSHIFT

LIST|NOLIST

Set the NOLIST option if a USE file echoed on the screen is not wanted.

Default: LIST

PRINT|NOPRINT (OpenVMS)

PRINT sends the source listing either to the file defined with the logical name SYSPRINT, or printer SYS$PRINT if SYSPRINT is not defined; NOPRINT does not. PRINT and NOPRINT are mutually exclusive options.

Limit: Valid only for PHDPDL.

SAVE CLEAR

Instructs PDL to clear the temporary save file for PDL statements. By default, PDL writes all statements entered to a temporary save file.

You may want to use this option before entering some other statements and then saving them with the SAVE statement. For more information about saving statements, see (p. 94).

VERIFY|NOVERIFY [DELETE][ERRORS]

The VERIFY option instructs PDL to interrupt processing, when necessary, with prompts for permission to proceed. When the VERIFY ERRORS option is used, PDL pauses when errors are found while processing statements from a file with the USE statement. You must press [Return] to continue after each error. When the VERIFY DELETE option is used, PDL requests authorization to delete an existing file and replace it with a new file of the same name.

The NOVERIFY option instructs PDL not to interrupt processing with prompts for permission to proceed. When the NOVERIFY ERRORS option is used, PDL does not pause when processing statements from a file with the USE statement. When the NOVERIFY DELETE option is used, PDL does not request authorization to delete an existing file.

Limit: DELETE is not valid in PHDPDL.

Defaults: NOVERIFY ERRORS; VERIFY DELETE

WARNINGS|NOWARNINGS

WARNINGS instructs PDL to issue warning messages when necessary. NOWARNINGS does not.

Default: WARNINGS

Discussion

The SET statement sets processing options that are in effect during a PDL or PHDPDL session. You can use it to change the default options that are in effect during a session. When the DEFAULT option is specified, it must be the only option in the SET statement. The SET DICTIONARY statement performs an implicit CANCEL statement.

SHOW DICTIONARY

Displays the name of the current dictionary.

Syntax

SHOW DICTIONARY

Discussion

The SHOW DICTIONARY statement displays the name of the current PDL dictionary. This setting can be changed with the SET DICTIONARY statement.

Example

If you enter

```
> SHOW DICTIONARY
```

the following is displayed on your terminal screen:

MPE/iX:	Current Dictionary is INVENTOR.DEVELOP.DOC.
OpenVMS:	PHDPDL: PATH$WRITER:[WRITER.MANAGER]VMSDICT.PHD PDL: Current Dictionary is PATH$WRITER:[WRITER.MANAGER]VMSDICT.PDC.
UNIX, Windows:	Current Dictionary is INVENTOR.

SYSTEM OPTIONS

Declares system-wide options and defaults for PowerHouse applications.

Syntax

SYSTEM OPTIONS option...

Options

SYSTEM OPTIONS Options	
`ALLOW|NOALLOW CENTURY`	`ASCII7|ASCII8`
`APPLICATION SECURITY|ASC`	`CENTURY INCLUDED|EXCLUDED`
`CHARACTER SET`	`DATE FORMAT`
`DATE SEPARATOR`	`DECIMAL`
`DEFAULT CENTURY`	`DEFAULT ENTRY AND FIND IN model SELECT IN model`
`DEFAULT TRANSACTION MODEL model`	`DICTIONARY OWNER`
`DICTIONARY SECURITY|DSC`	`FLOAT`
`FORCE|NOFORCE CENTURY`	`GENERIC RETRIEVAL CHARACTER`
`INPUT CENTURY`	`MESSAGE SUBSTITUTION CHARACTER`
`MULTILINE HEADING CHARACTER`	`NULL|MISSING VALUE CHARACTER`
`NULLSEPARATOR|NONULLSEPARATOR`	`PATTERN`
`PICTURE SUBSTITUTION CHARACTER`	`PORT`
`RELEASE`	`SHIFT`
`SHOW|NOSHOW`	`SPECIAL NAME CHARACTERS`
`SYSMONTHS`	`TITLE`
`VERSION`	

ALLOW|NOALLOW CENTURY

ALLOW CENTURY specifies that the user can enter a century on date fields even though only a two-digit year is specified in the date format. The option applies to century-included date fields with a two-digit year format.

When ALLOW CENTURY is specified, date fields on QUICK screens become horizontally scrolling fields. The user can then enter the date including the century in the same space as the date without the century. The century is not displayed after input.

The "more data" character, which identifies horizontally scrolling fields, is displayed outside the data area. This could result in compilation warnings if the "more data" character is displayed on top of the screen background.

Limit: Valid only for century-included dates.

Default: NOALLOW CENTURY

ASCII7|ASCII8

ASCII7 uses the seven-bit ASCII sort, upshift, and downshift tables. ASCII8 uses the eight-bit ASCII sort, upshift, and downshift tables. ASCII7 and ASCII8 are mutually exclusive options.

Default: ASCII7

APPLICATION SECURITY ID METHOD id-method
ASC ID METHOD id-method

Establishes the ID method that PowerHouse uses to put users in an application security class (ASC).

id-method

MPE/iX:	Either LOGONID, WEBLOGONID, or NONE.
OpenVMS:	Either LOGONID, PASSWORD, PORTID, UIC, WEBLOGONID, or NONE.
UNIX:	Either LOGONID, UIC, WEBLOGONID, or NONE.
Windows:	Either LOGONID, WEBLOGONID, or NONE.

Default: NONE

CENTURY {INCLUDED|EXCLUDED}

Specifies whether dates include a four-digit year or a two-digit year.

Default: CENTURY INCLUDED

CHARACTER SET REPLACED BY option...
CHARACTER SET {ENGLISH|FRENCH|GERMAN} [option]...

Establishes the language of the character set used by PowerHouse. The possible values are ENGLISH, FRENCH, GERMAN, or REPLACED BY. The REPLACED BY option allows creation of a unique character set. If this option is selected, all 128 characters must be entered (if seven-bit ASCII is selected) or all 256 characters (if eight-bit ASCII is specified).

In combination with the seven or eight-bit ASCII specification, this option determines what collating sequence PowerHouse uses for sorting and comparing alphanumeric values. This option also determines the correspondences PowerHouse uses to upshift and downshift special letters (for example, lowercase é to uppercase É in French).

For more information about character sets, see (p. 110).

Default: ENGLISH

Character set options are sort-spec, UPSHIFT upspec, and DOWNSHIFT down-spec.

[sort-spec]

Sort-spec establishes the collating sequence used by PowerHouse to sort and compare values.

UPSHIFT up-spec

Used to shift entered value of character to uppercase.

Limit: 60 characters

DOWNSHIFT down-spec

Used to shift entered value of character to lowercase (DOWNSHIFT). Using same case values avoids ambiguous entries. Otherwise, uppercase and lowercase letters are treated as distinct.

Limit: 60 characters

The syntax for sort-spec, up-spec, and down-spec takes the general form:

ASCII|HEXADECIMAL string [ASCII| HEXADECIMAL string]...

If the REPLACED BY option is used, then the sort-spec must contain 128 characters for seven-bit ASCII or 256 characters for eight-bit ASCII, depending on which was specified in the current PDL session or in the data dictionary. If neither 128 nor 256 is specified, then a syntax error is issued. If REPLACED BY is used, the only options available are ASCII string or HEXADECIMAL string.

If using hexadecimal values, ensure that CHARACTER SET REPLACED BY HEXADECIMAL is specified in both PDC and PHD dictionaries. This ensures consistent behavior in QSHOW when generating hex values.

The UPSHIFT and DOWNSHIFT options determine the correspondences used when shifting the case of special letters. Any of these options can be applied to the character set selected, or to the user-defined character set. That is, begin by specifying the English, French, or German character sets and then make modifications to the character set selected by specifying any combination of a sort sequence, upshift, and downshift specifications. If a base language is not specified, English is used.

DATE FORMAT date-format

Sets the default date format for all date elements. Unlike the date format in an ELEMENT statement, the system-wide date format must include the day, month, and year. Date values can be entered either with or without separator characters. Dates can be displayed using any of the following date formats.

These examples use the slash (/) as a date separator, and a default century of 20.

date-format	Example	date-format	Example
YYMMDD	01/05/23	YYMMMDD	01/MAY/23
YYYYMMDD	2001/05/23	YYYYMMMDD	2001/MAY/23
MMDDYY	05/23/01	MMMDDYY	MAY/23/01
MMDDYYYY	05/23/2001	MMMDDYYYY	MAY/23/2001
DDMMYY	23/05/01	DDMMMYY	23/MAY/01
DDMMYYYY	23/05/2001	DDMMMYYYY	23/MAY/2001

The default date format can be overridden in other PowerHouse components. For more information, see the CENTURY INCLUDED option of the SYSTEM OPTIONS statement on (p. 101) and DEFAULT CENTURY option of the SYSTEM OPTIONS statement on (p. 102).

Default: YYMMDD for two-digit years if the CENTURY EXCLUDED option of the SYSTEM OPTIONS statement is specified; otherwise, the default is YYYYMMDD.

DATE SEPARATOR char

Assigns the default character that separates different portions of a date. For example, a system-wide date format of YYYYMMDD and a system-wide date separator of a hyphen (-) produce dates such as 2006-08-21. This option can be used alone or with the DATE FORMAT option.

Default: A slash (/)

DECIMAL char

Sets the character to be used as a decimal marker when entering and displaying numbers. The decimal character must be either a comma or a period. The decimal character affects all default pictures for element and usage definitions.

When the decimal character is set to a period, the comma is used to delimit 1000s. If a comma is used as the decimal character, the delimiter is a space.

Default: A period (.)

DEFAULT CENTURY n

Establishes the century to be used as the default for dates with four-digit years.

The default century is retrieved from the dictionary at run-time. You can change the default century by changing the option and recompiling the dictionary.

Default: 19

DEFAULT ENTRY [AND FIND] [IN] CONCURRENCY|CONSISTENCY|OPTIMISTIC SELECT [IN] CONCURRENCY|CONSISTENCY|OPTIMISTIC

Specifies what default models are used for screens using the DUAL model. Screen designers can override these defaults by using the TRANSACTION MODEL option of the SCREEN statement.

If the same model is specified for both Select and Entry/Find modes, a warning message is issued.

Defaults:

ENTRY AND FIND IN CONSISTENCY

SELECT IN CONCURRENCY

DEFAULT TRANSACTION MODEL CONCURRENCY|CONSISTENCY|DUAL|OPTIMISTIC

Specifies the default transaction model used by QUICK screens. Screen designers can override this default by using the TRANSACTION MODEL option of the SCREEN statement.

Default: CONCURRENCY

CONCURRENCY

Specifies that the Concurrency transaction model is used.

CONSISTENCY

Specifies that the Consistency transaction model is used.

DUAL

Specifies that the Dual transaction model is used. The default models used for the Entry/Find and Select modes for the Dual model are:

Defaults:

ENTRY AND FIND IN CONSISTENCY

SELECT IN CONCURRENCY

OPTIMISTIC

Specifies that the Optimistic transaction model is used.

DICTIONARY OWNER string (OpenVMS)

Defines the owner of the dictionary. The owner can always change the dictionary, no matter what dictionary security class is in effect.

string

Specifies the logon ID of the user who is to be the dictionary owner. Ensure that you specify a valid logon ID to avoid the possibility of restricting all users from the dictionary.

Default: The creator of the dictionary.

DICTIONARY SECURITY ID METHOD id-method (OpenVMS) DSC ID METHOD id-method

Establishes the ID method that PowerHouse uses to put users in a dictionary security class (DSC). You can only define one ID method at a time for any given dictionary.

The syntax is parsed in PDL but only used in PHDPDL and the PhD Screen System.

id-method

One of LOGONID, NONE, PASSWORD, PORTID, or UIC.

Default: NONE. When the ID method is set to NONE, all PhD Screen System users have full access to the dictionary.

FLOAT {IEEE|NONIEEE} (MPE/iX)

Native Mode PowerHouse supports both IEEE and NONIEEE FLOAT formats.

Default: IEEE

FORCE|NOFORCE CENTURY

FORCE CENTURY specifies that the user must enter a century on all century-included date fields. The option applies to century-included dates with two or four-digit year formats.

Limit: Valid only for century-included dates.

Default: NOFORCE CENTURY

GENERIC RETRIEVAL CHARACTER char

Establishes the character used for partial-index retrieval in QTP and QUIZ CHOOSE statements. In QUICK, the generic retrieval character specified in the QKGO file is used instead of the generic retrieval character in the dictionary.

Default: An at-sign (@)

INPUT CENTURY century FROM [YEAR] year

Establishes an input century window. The window is used to determine which century value will be added to a century-included date when only a two-digit year is input.

This option applies to century-included dates input in QUICK, QUIZ, and QTP.

The INPUT CENTURY values are not compiled into PowerHouse applications. To change the INPUT CENTURY values, you change the PDL option and recompile the dictionary. It is not necessary to recompile your application to change the INPUT CENTURY window.

If the INPUT CENTURY is not specified, the DEFAULT CENTURY is used as the default input century.

century

The century to be used if the year of the entered date is equal to or greater than the FROM year. Entered dates with a year that is less than the FROM year will use the value of century + 1.

Limits: 1 to 99

year

The lower limit of the input century window. The upper limit is always 99.

Limits: 0 to 99

MESSAGE SUBSTITUTION CHARACTER char

Establishes the character that marks where PowerHouse puts variable parts of messages into the standard PowerHouse messages (for example, the file name in a "File exists?" message). If you change this character, you must change the substitution character in existing message files to the new character.

Default: A caret (^)

MULTILINE HEADING CHARACTER char

Establishes the character that indicates where QUIZ splits a heading onto another line. If you change this character, you must change the strings in all HEADING options in your dictionary definition and reload the dictionary. You must also change the HEADING options in all your QUIZ reports.

Default: A caret (^)

NULL|MISSING VALUE CHARACTER char

Establishes what character is to be displayed when a defined or temporary item is null (missing), or when an item in a table is displayed but not retrieved from the database.

Default: A blank

NULLSEPARATOR|NONULLSEPARATOR

NULLSEPARATOR specifies that all dates are to be displayed without a separator. This allows display of century-included dates in the same space as century-excluded dates.

The DATE SEPARATOR option is used for display formatting if NULLSEPARATOR is not used, or is canceled by the NONULLSEPARATOR option.

The DATE SEPARATOR option may be used during input. If NULLSEPARATOR is specified, the value is redisplayed after formatting without the separator.

Default: NONULLSEPARATOR

PATTERN [metachar]... [RESERVED [CHARACTERS]string]

Specifies a set of special characters (metacharacters) that PowerHouse uses during pattern matching. You would change the metacharacters if the PowerHouse default character conflicts with a common character in the local language, or if the character isn't supported by the platform on which your application runs.

metachar

Represents a PDL keyword describing a pattern matching metacharacter. It is a character that describes a class of characters or something about the pattern rather than simply matching itself. A metacharacter equates a metacharacter name and function with a character. For example, to change the ANY metacharacter from the default question mark (?) to percent sign (%), enter

```
> SYSTEM OPTIONS &
>   PATTERN ANY "%"
```

The following table lists the metacharacters defined in PowerHouse.

Metacharacter	PDL Keyword	Function
^	ALPHA	Matches any single uppercase or lowercase alphabetic character.
?	ANY	Matches any single character (alphabetic, numeric, or special) including blanks.
#	DIGIT	Matches any single numeric digit.
!	ESCAPE	Signifies that the character immediately following the escape character is to be interpreted as a regular character rather than as a special character, reserved character, or metacharacter. For example, the pattern, `"^^^^!?"`, requires a question mark as the fifth character. The escape character cannot be used with alphabetic or numeric characters (except !0) and it has no effect on other characters that aren't metacharacters.
( )	LEFTP, RIGHTP	Together, these metacharacters determine the order in which the characters in a pattern are to be interpreted.
!0	NULL	Matches a no-character or null entry. The NULL metacharacter is always the active escape character followed by 0. The escape character is normally an exclamation mark.

Metacharacter	PDL Keyword	Function
\	NOT	Disallows the character that immediately follows. For example, the pattern, "###\0", accepts any three-digit number followed by any digit other than 0.
<	OPTIONAL	Matches zero or one occurrence of the immediately preceding character (or pattern string in parentheses). For example, the pattern, "a<", accepts either a null value or the character, "a".
*	OPTREP	Matches zero or more occurrences of the immediately preceding character (or pattern string in parentheses). For example, the pattern, "a*", accepts either a null value or a string containing one or more occurrences of the character, "a".
|	OR	Matches a single occurrence of either the preceding or the following character. For example, the pattern, "a|b", matches either the character, "a", or the character, "b". You can match groups of characters (strings) by enclosing the group in parentheses.
>	REPEAT	Matches one or more occurrences of the immediately preceding character (or pattern string in parentheses). For example, the pattern, "a>", doesn't accept a null value, but it does accept a string containing one or more occurrences of the character, "a".
@	WILD	Matches zero or more characters (alphabetic, numeric, or special).

RESERVED [[CHARACTERS] string]

Changes the list of PowerHouse reserved metacharacters.

By default, PDC dictionaries reserve a seven character string for future expansion of pattern matches; PHD dictionaries reserve a six character string. They are:

Platform	Dictionary Type	Default Reserved Characters
MPE/iX, OpenVMS, UNIX, Windows	PDC	[ ] : = ; _ &
OpenVMS	PHD	[ ] : = ; &

Replacement characters must be matched one-for-one with the list of special characters. The first character you specify will replace the left bracket ([), the second character will replace the right bracket (]), and so on.

For example, the following entry in a PDC dictionary

```
> SYSTEM OPTIONS PATTERN &
>     RESERVED CHARACTERS "$%:=;-&"
```

changes the list of PowerHouse reserved metacharacters to

```
$  %  :  =  ; - &
```

OpenVMS: PowerHouse accepts a minimum of six characters and a maximum of seven characters in reserved character strings. If five character strings are used, the compiler converts them to six character strings. For more information about how five character strings are converted, see (p. 111).

Although the underscore (_) is no longer considered a reserved character for PDC dictionaries, it is included as a default reserved character for backwards compatibility. Users that want to allow the underscore in patterns without having to use an escape character can use the six character string, "[ ] : = ; &".

PICTURE SUBSTITUTION CHARACTER char

Establishes the substitution character for PICTURE options in QUICK and QUIZ. If you change this character, you must change your dictionary definition and any QUICK screens and QUIZ reports that have PICTURE options.

Default: A caret (^)

PORT port# [TO port#] [TERMINAL] [TYPE] string (MPE/iX)
PORT ttyname [TERMINAL] [TYPE] string (UNIX)

Assigns a terminal type to a port.

port# [TO port#] (MPE/iX)

A logical device number (or range of numbers) of the terminal ports to be used for QUICK applications.

ttyname (UNIX)

Indicates the pathname for a terminal; for example, /dev/ttydqp2.

[TERMINAL] [TYPE]

Specify one or both of these keywords prior to specifying the terminal type used. These keywords are for documentation only.

string

A string identifying the termtype. A name used to identify a terminal's manufacturer and model to PowerHouse, for example, HP2392. Using the port option to specify the terminal type prevents QUICK from prompting the user for the terminal type. (For information on supported terminal names, see the supplied terminfo file.)

RELEASE n [VERSION n]

PowerHouse checks the release number in the dictionary before it runs a compiled report, run, or screen. PowerHouse displays an error message if the current release number in the dictionary is different from the release number when the report, run, or screen was compiled. This prevents PowerHouse from producing erroneous results due to a changed dictionary.

Limit: 0 to 255

VERSION n

The VERSION option sets a number that supplements the release number. The version number is a documentation aid and is not checked by the PowerHouse components at execution time.

Note: The VERSION suboption also exists as the VERSION option of the SYSTEM OPTIONS statement. See (p. 107).

Limit: 0 to 255

SHIFT DOWNSHIFT|NOSHIFT|UPSHIFT

Specifies the system-wide access to mixed, lowercase or uppercase identifiers. The SHIFT option is valid for both PDC and PHD dictionaries.

Except for PHDPDL and the PhD Screen System, the DOWNSHIFT, NOSHIFT, and UPSHIFT options of the SET statement and the **downshift, upshift,** and **noshift** program parameters can be used to override the SHIFT option of the SYSTEM OPTIONS statement. The DOWNSHIFT, NOSHIFT, and UPSHIFT options of the SET statement are not available in PHDPDL. The **downshift, upshift,** and **noshift** program parameters are ignored in both PHDPDL and the PhD Screen System.

Limit: SHIFT does not affect the syntax name shifting in the PhD Screen System or PHDPDL. The PhD Screen System and PHDPDL use the default, UPSHIFT, except for the PASSWORD and USERID options of the FILE and DATABASE statements.

Default: UPSHIFT

SHOW|NOSHOW

SHOW places no restrictions on the user's ability to report dictionary contents. NOSHOW ensures that PowerHouse users cannot report the contents of the dictionary with QSHOW or with the SHOW statement in QDESIGN, QTP, and QUIZ.

OpenVMS: NOSHOW does not restrict dictionary owners or Dictionary Managers from reporting the contents of PHDPDL dictionaries.

Default: SHOW

SPECIAL NAME CHARACTERS string

Establishes the set of special characters that can be used in entity names in the data dictionary, for example, file and element names.

Limit: Spaces and nonprinting characters cannot be used in names.

The following group of special characters and control characters are not allowed in entity names since these characters are part of the syntax of PowerHouse:

ampersand	&	semicolon	;
comma	,	tab	
carriage return		space	

The special characters that are safe to use are:

exclamation mark	!	number sign	#
close single quote	’	tilde	~
backslash	\	vertical bar	|
percent sign	%	dollar sign	$
brackets	{ }	underscore	_

The special characters that should be used with caution are:

caret	^	asterisk	*
parentheses	( )	minus sign	-
plus sign	+	colon	:
square brackets	[ ]	double quotation marks	"
angle brackets	<>	open single quote	’
forward slash	/	equal sign	=
decimal point	.	question mark	?
at sign	@		

You must use caution with these characters because they serve not only as characters in dictionary names, but also in mathematical operations and PowerHouse functions. This can cause problems in PowerHouse components. For example, if a hyphen is allowed in dictionary names, the QUIZ statement

```
> DEFINE TOTAL NUM*6 = INVOICE-DISCOUNT
```

could be ambiguous. It could be interpreted as TOTAL being equal to the value of the item, INVOICE-DISCOUNT, or as TOTAL being equal to the result of subtracting the value of DISCOUNT from the value of INVOICE. If no item named INVOICE-DISCOUNT exists there is no ambiguity. But in most cases, if these special characters are used in dictionary names, users of the PowerHouse components should precede these characters with a space when using them in mathematical operations or functions. The statement

```
> DEFINE TOTAL NUM*6 = INVOICE - DISCOUNT
```

cannot be misinterpreted.

In a similar way, if a question mark (?) is used as the last character in a dictionary name, some confusion could result with the use of a question mark to find out the acceptable entries at any point in a QUIZ, QDESIGN, QTP, PDL, or QSHOW session.

The at sign (@) character can cause problems if it is used as the last character in a dictionary name (for example, INVOICE@). When partial retrieval is used in the PhD Screen System or QUICK screens, the @ character is interpreted as initiating partial retrieval, not as the last character in a name.

If the SPECIAL NAME CHARACTERS option is not used, the default characters are:

hyphen	-	percent sign	%
underscore	_	number sign	#
open single quote	’	dollar sign	$

SYSMONTHS string

Specifies a 36-character string consisting of 12 three-character abbreviations for the month names.

The abbreviations are used in QUIZ reports and QUICK screens. For example, the following entry changes the abbreviations from English to French.

```
> SYSTEM OPTIONS SYSMONTH &
>   "JANFEVMARAVRMAIJUNJULAOUSEPOCTNOVDEC"
```

Note: There are no French three-character abbreviations for the months of June and July. The preceding example uses the abbreviations, "JUN" and "JUL", respectively, for June and July.

Default string: JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC

TITLE string

Specifies the default report title used by QUIZ and QSHOW, and the title returned by the PowerHouse system function SYSNAME.

Limit: 40 characters

VERSION n

The VERSION option sets a number that supplements the release number. The version number is a documentation aid and is not checked by the PowerHouse components at execution time.

Note: The VERSION option also exists as a suboption of the RELEASE option. See (p. 107).

Limit: 0 to 255

Discussion

The SYSTEM OPTIONS statement sets system-wide options. System-wide options influence the default used by the PowerHouse components, including PDL. Although you can enter only one SYSTEM OPTIONS statement, you can include as many options as you like on that one statement.

The system-wide options govern the following:

- the character set used
- the default appearance of dates
- special characters used for generic retrieval, picture substitution, message substitution, or as the multi-line heading character
- the character used as a decimal point
- the special characters allowed in entity names
- current ASC ID methods
- current DSC ID methods (**OpenVMS:** PHDPDL)
- dictionary owner (**OpenVMS:** PHDPDL)
- the release and version of the dictionary
- the characters used as metacharacters in pattern matching

They also govern the following:

- the character string consisting of abbreviations for month names
- the default report title used by QUIZ and QSHOW
- the user's ability to report dictionary contents
- assigning a terminal type to a port

The eight date options provide flexibility when entering, displaying, and storing dates. These options are: ALLOW|NOALLOW CENTURY, FORCE|NOFORCE CENTURY, INPUT CENTURY, NULLSEPARATOR|NONULLSEPARATOR, CENTURY INCLUDED|EXCLUDED, FORMAT, SEPARATOR, and DEFAULT CENTURY.

The date options specified on ELEMENT and USAGE statements take precedence over equivalent or conflicting options specified on the SYSTEM OPTIONS statement. The DEFAULT CENTURY and INPUT CENTURY options apply system-wide, and can be used only with the SYSTEM OPTIONS statement.

Many of the SYSTEM OPTIONS can be overridden in QUICK, QTP, and QUIZ.

Character Set

The CHARACTER SET option allows you to use one of three pre-established character sets or to establish your own. It works with the seven or eight-bit ASCII specification to establish internal tables that PowerHouse uses for sorting, upshifting, and downshifting item values. PowerHouse supports different sorting (collating) sequences for English, French, and German. As well, you can define your sort sequence. The sort, upshift, and downshift specifications of the CHARACTER SET option can be applied to the character set selected, or to the user-defined character set. If a base language is not specified, English is used.

Modifying the Sort Sequence

The sort specification takes the general form

ASCII|HEXADECIMAL string [ASCII|HEXADECIMAL string]...

The ASCII keyword is followed by a string of ASCII characters. The HEXADECIMAL keyword is followed by a string of a series of characters in hexadecimal notation. The first character in the string is the starting point for the modified sorting sequence. The rest of the string gives the characters that are to follow the first character. For example, special characters fall in four different places in the ASCII sorting sequence. If you want all special characters to precede the digit 0 in sorting, the CHARACTER SET option looks like this:

```
> SYSTEM OPTIONS &
> CHARACTER SET ASCII "/:;<=>?@[\}^_`{|}~"
```

The first character in the string is the starting point for the modified sort sequence. The following characters in the string give the new sort sequence. In this example the new sort sequence specified that the characters :;>=>?@[\}^_‘{|}~ are to sort after the slash (and thus, before the digit 0).

The HEXADECIMAL option works the same way as the ASCII option. The above example can be achieved using the following statement:

```
> SYSTEM OPTIONS &
> CHARACTER SET HEXADECIMAL &
> "2F3A3B3C3D3E3F405B5C5D5E5F607B7C7D7E"
```

The HEXADECIMAL option is useful when specifying the sort sequence of unprintable ASCII characters.

Defining Shift Specifications

As well as specifying a non-standard sort sequence, you can specify your own upshift and downshift tables. PowerHouse uses the shift tables to change the case of letters. It does this with the

- UPSHIFT and DOWNSHIFT functions
- UPSHIFT and DOWNSHIFT options of the QUIZ and QTP CHOOSE, EDIT, and DEFINE statements
- UPSHIFT and DOWNSHIFT options of the QTP GLOBAL TEMPORARY statement
- UPSHIFT and DOWNSHIFT options of the QDESIGN FIELD statement

The shift tables are also used in pattern matching to determine valid alphabetic characters when the ALPHA metacharacter is used

The purpose of the shift tables is to allow you to specify the correct shift character for a nonstandard ASCII character. You can specify the upshift and downshift correspondences in ASCII or hexadecimal notation. If you enter either the UPSHIFT or DOWNSHIFT option, you erase the default shift tables and replace them with new shift tables. The new shift tables contain the standard equivalencies of upper and lowercase letters for the 26 letters of the English alphabet, plus the equivalencies specified in the UPSHIFT or DOWNSHIFT option. PowerHouse always has the shift information for the 26 letters of the English alphabet.

PowerHouse shifts each odd character (or pair of hexadecimal digits) to the character (pair of digits) that follows it. For example, if you have

```
UPSHIFT ascii "éÉñÑ"
```

PowerHouse will change the lowercase accented letters to the correct uppercase accented letters.

Handling Reserved Characters (OpenVMS)

To allow backwards compatibility, both the PDL and PHDPDL compilers can parse reserved character strings containing five characters. When the compiler encounters a five character string, it issues a warning that the five character string is obsolete. At least six characters are required on input. The compiler accepts the five character string and appends an additional character to create a six character string. If a user-defined Kanji metacharacter exists, it is inserted into the five character string after the second reserved character, otherwise the colon (:) is used. The resulting six character string is accepted as input. For example, the five character string

```
"[ ] = ; &"
```

changes to the six character string

```
"[ ] : = ; &"
```

In QSHOW, the SHOW SYSTEM statement displays the 6 character string derived from the five character string.

For more information on the reserved characters, see the RESERVED option on (p. 106).

Examples

The SYSTEM OPTIONS statement in the following example specifies the format and default separator for the date:

```
> CREATE DICTIONARY PHD
> SYSTEM OPTIONS &
>      DATE FORMAT DDMMMYYYY &
>      DATE SEPARATOR "*" &
>      CENTURY INCLUDED
.
.
.
```

An example of a date format in this system would be 09*JAN*1999.

The following SYSTEM OPTIONS statement uses all the year 2000 features. The DEFAULT CENTURY is not specified, but its value is, by default, 19. The processing applies to all relevant screens, reports and runs in the application. To be more selective in the use of the options, use them on specific elements, usages, or statements in screens, reports or runs.

```
> SYSTEM OPTIONS &
>   ALLOW CENTURY &
>   FORCE CENTURY &
>   INPUT CENTURY 19 FROM YEAR 80
>   NULLSEPARATOR
```

In the following example, the input century window is from 85 to 99. All dates with a year from 85 to 99 will have a default input century of 19. All dates with a year less than 85 will have a default input century of 20.

```
> SYSTEM OPTIONS &
>   INPUT CENTURY 19 FROM YEAR 85
```

In the next example, the input century window is from 0 to 99. All input dates will have a default input century of 20.

```
> SYSTEM OPTIONS &
>   INPUT CENTURY 20 FROM YEAR 0
```

TRANSACTION

Defines transactions used for relational files. Override the default transaction characteristics that PowerHouse uses.

Syntax

TRANSACTION name [option]...

name

Any valid PowerHouse name. The transaction name must be unique within the dictionary.

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters that are defined in the dictionary.

Options

TRANSACTION Options		
CONSTRAINTS	COMMIT|NOCOMMIT	DATABASE
isolation-level	PRIORITY	READ ONLY|READ WRITE
RESERVING	WAIT|NOWAIT	

CONSTRAINTS [ALL|{CHECK|REFERENTIAL|UNIQUE} [,{CHECK|REFERENTIAL|UNIQUE}]...] DEFERRED

Allows specific types of constraints to be deferred while others are checked immediately.

Limit: Valid only for ALLBASE/SQL transactions.

Default: The default constraint type is ALL. If constraints are not deferred, they are checked every time a value is inserted, altered or deleted.

{COMMIT [ON] automatic-commit-point}|NOCOMMIT

The COMMIT option is used to indicate the default points at which automatic commits are executed by QUICK.

The NOCOMMIT option indicates that QDESIGN does not generate automatic commit actions. This does not affect the CONTINUE option.

automatic-commitpoint

The automatic-commit-points are UPDATE, MODE, NEXT PRIMARY, and EXIT.

Determines the points at which an automatic commit for the COMMIT option, or an automatic save for the CONTINUE option occur during screen processing.

Limit: Any automatic-commit-points specified on the PDL TRANSACTION statement are ignored by QUIZ and QTP.

For more information about automatic commit points, see Chapter 3, "QDESIGN Statements", in the *QDESIGN Reference* book.

DATABASE name isolation-level [DATABASE name isolation-level]...

Allows the specification of an isolation level per database. The database name must refer to the logical name as defined in an earlier DATABASE statement. The isolation-level is as described under the isolation-level option.

isolation-level

Lets the designer specify the degree to which this transaction is to be protected from the effects of concurrent transactions.

The isolation levels are listed below from lowest to highest:

READ UNCOMMITTED

A very low level of isolation that allows a transaction to see all changes made by other transactions, whether committed or not.

Also known as a "dirty read".

READ COMMITTED

A transaction can read any data that has been committed by any transaction as of the time the read is done.

STABLE CURSOR

While a transaction has addressability to a record (that is, has just fetched it), no other transaction is allowed to change or delete it.

REPEATABLE READ

Any data that has been read during a transaction can be re-read at any point within that transaction with identical results.

PHANTOM PROTECTION

A transaction does not see new records, or "phantoms", that did not exist when the transaction started.

SERIALIZABLE

The results of the execution of a group of concurrent transactions must be the same as would be achieved executing those same transactions serially in some order.

Default: READ COMMITTED if the string "QUERY" (case-insensitive) appears anywhere in the transaction's name. Otherwise, REPEATABLE READ is the default.

The support available for the various isolation level options depends on the support provided by the underlying database software.

If a database doesn't support a specified isolation level, PowerHouse uses the next available higher isolation level without issuing a run-time warning. If a higher level is unavailable, PowerHouse uses the highest available lower level and issues a run-time warning.

PRIORITY n

Lets you specify the transaction priority for an ALLBASE/SQL transaction.

n

An integer in the range 0 to 255.

Limit: Valid only for ALLBASE/SQL.

READ ONLY|READ WRITE

Determines the type of activities that can be performed by this transaction. It also affects what type of transaction is started in the underlying database system.

Default: READ ONLY if the string "QUERY" (case-insensitive) appears anywhere in the transaction's name. Otherwise, READ WRITE.

RESERVING FOR [EXCLUSIVE|PROTECTED|SHARED] READ|WRITE {table [IN database]}...

Lets you specify database specific reserving on a table-by-table basis for a particular transaction.

ORACLE synonyms may be used for table-names. For more information about how PowerHouse uses ORACLE synonyms, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

READ only lets you read data from the reserved tables; WRITE lets you insert, update, or delete data in the table.

Limit: The RESERVING option is ignored by QUIZ and QTP. It is not supported for SYBASE databases.

Default: READ for read-only transactions. WRITE for read/write transactions. The defaults are determined from the READ ONLY| READ WRITE option.

SHARED|PROTECTED|EXCLUSIVE

SHARED lets others work with the same table(s). PROTECTED lets others read the table you are using; they cannot have write access. EXCLUSIVE prevents others from reading records from the table(s) included in your transaction.

Default: SHARED

WAIT|NOWAIT

Allows you to specify whether the transaction should wait in the case of lock or resource conflicts.

If NOWAIT is specified, then a resource or lock conflict causes the transaction to end with an error condition.

If WAIT is specified, then a resource or lock conflict causes the transaction to wait, subject to other system parameter and timeout settings.

Default: Determined at run-time by the **dbwait** program parameter or resource file statement. If no values have been specified for **dbwait**, WAIT is used.

Discussion

The TRANSACTION statement is used to define a transaction and its attributes. Only QDESIGN uses the COMMIT automatic-commit-point options and the RESERVING option.

The PowerHouse Default Transactions

The PowerHouse default transactions are:

Transaction	Access	Isolation-level	PowerHouse Component
Consistency	read/write	SERIALIZABLE	QDESIGN, QTP
Query	read-only	READ COMMITTED	QDESIGN, QUIZ, QTP
Update (DB2, Oracle Rdb, SYBASE)	read/write	REPEATABLE READ	QDESIGN, QTP
Update (ALLBASE/SQL, ODBC, ORACLE)	read/write	READ COMMITTED	QDESIGN, QTP

Overriding the Default Transactions

If you are satisfied with the attributes of the PowerHouse default transactions, then there is no need to define TRANSACTION statements in the dictionary. These default transactions can still be referenced by name on the FILE and CURSOR statements, on SQL DML statements/verbs, or in control structures without being explicitly defined each time.

To change the attributes of the PowerHouse default transactions, you include a TRANSACTION statement with the name of the default transaction along with the attributes to be changed. All unspecified attributes remain unchanged. For example, the following changes the isolation level of the Query transaction to SERIALIZABLE.

```
> TRANSACTION QUERY SERIALIZABLE
```

Only the isolation level is affected; all other attributes are retained.

Customizing Transactions

You can define your own transactions or modify the attributes of the PowerHouse default transactions with the TRANSACTION statement.

QDESIGN

In QDESIGN, the attributes for a transaction are determined as follows:

1. The attributes are set to default values.
2. If the transaction is defined in the dictionary, then the attributes specified in the dictionary are applied and override any default attributes.
3. If there is a transaction defined on the screen, then the attributes specified on the QDESIGN TRANSACTION statement are applied and override any attributes defined previously.

QUIZ

The behavior of the transaction that QUIZ uses may be customized. In QUIZ, the attributes for the Query transaction are determined as follows:

1. QUIZ sets the attributes by looking in the dictionary for a transaction named QUIZ_QUERY.
2. If a QUIZ_QUERY transaction has not been defined in PDL, then QUIZ sets the attributes by looking in the dictionary for a transaction named QUERY.
3. If there is no QUIZ_QUERY or QUERY transaction defined in the dictionary, then the transaction name defaults to QUERY and the attributes are set to the default values specified for the options of the PDL TRANSACTION statement.

QTP

The behavior of the transaction that QTP uses may be customized. In QTP, the attributes for the Query, Consistency, and Update transactions are determined as follows:

1. QTP sets the attributes by looking in the dictionary for a transaction named QTP_QUERY.
2. If a QTP_QUERY transaction has not been defined in PDL, then QTP sets the attributes by looking in the dictionary for a transaction named QUERY.
3. If there is no QTP_QUERY or QUERY transaction defined in the dictionary, then the transaction name defaults to QUERY. Its attributes are set to the default values specified for the options of the TRANSACTION statement described above.

   The same three-step process applies for determining attributes for the Consistency and Update transactions, in which case QTP looks for the QTP_CONSISTENCY, CONSISTENCY, QTP_UPDATE, and UPDATE transactions.

QTP does not look in the dictionary for a transaction with an appended open number. For example, QTP will not look in the dictionary for a transaction named QUERY_02.

The Effect of the PDL TRANSACTION Statement on QDESIGN

Options specified on the TRANSACTION statement in PDL are used in QDESIGN for transactions of the same name.

Reserving on a Table-by-Table Basis

Tables to be reserved are specified using the "table IN database" of the RESERVING FOR option, on the TRANSACTION statement. This allows you to include tables in the RESERVING list that are not defined on the screen where the transaction is started. For example, you can reserve tables that are used in a group of screens.

Note: This applies to QUICK only.

If a transaction is started automatically as a result of an access or write to a database, those tables in the reserving list from a different database will not be reserved. If a transaction is started explicitly using the START TRANSACTION verb, then all tables from all databases in the reserving list will be reserved.

Reserving options are specified using keywords that match options available in the underlying database software.

The RESERVING option is especially useful for high contention applications, such as those which involve reservation of inventory or seats. It also provides a means of ensuring deadlock-free transactions, since all tables are locked before the transaction does any other work. (Consider, however, the impact that any table locking has on concurrent database users—there is always a trade-off. Remember also that table locks cannot be released until the transaction terminates.) The RESERVING option also provides a means of guaranteeing successful serializable updates to the reserved tables.

USAGE

Declares a usage.

Syntax

USAGE name element-type [SIZE] n [option]...

name

Assigns a name to a new usage or identifies a usage that has already been defined in the dictionary.

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters defined in the dictionary by the SPECIAL NAME CHARACTERS option of the SYSTEM OPTIONS statement.

element-type

Specifies the element type that will be used for elements created from this usage. The type must be one of the following:

- CHARACTER or C
- DATE or D
- NUMERIC or N

[SIZE] n

Specifies the number of characters or digits allowed in elements created from this usage. The keyword SIZE is optional, but you must specify a number. If you enter SIZE 0 for the usage, then you must enter the element size on the ELEMENT statement for each element that uses this usage.

While most usages specify an element size (for example, the usage NAME has an element size of 20), sometimes you may want to let different elements that use the usage have different sizes. You can do this by entering 0 when declaring the size of a usage in the USAGE statement. The default item size will specify a minimum size, which is expanded when the user enters an element size. The standard usages ID and NUMERIC-ID use this feature.

When establishing a usage with an element size of 0, you can't anticipate the correct number of substitution characters (^) to use in the picture; that depends on the element size in the ELEMENT statement. However, you can declare pictures for numeric elements that PDL can adjust for all element sizes chosen by the user. Enter a picture that is large enough for all reasonable element sizes that users might choose. When a size is specified, extra substitution characters and any characters in between them on the left are removed.

Limit: Dates must have a size of between 6 and 16. Numeric types have a limit of up to 31 except for ZONED and PACKED datatypes on MPE/iX which have a limit of 28. Characters have a limit of 2047.

Options

Most of the optional attributes of element usages are identical to those of elements. For explanations of the following attributes, see (p. 44) to (p. 53).

USAGE options

`ALLOW|NOALLOW CENTURY`	`BWZ`
`DECIMAL n`	`DOWNSHIFT`
`FILL char`	`FLOAT char`
`FORCE|NOFORCE CENTURY`	`FORMAT date-format`
`INITIAL value`	`INPUT SCALE n`
`LEADING SIGN char`	`NULLSEPARATOR|NONULLSEPARATOR`
`OCCURS`	`OUTPUT SCALE n`

USAGE options	
`PATTERN string`	`PICTURE string`
`SEPARATOR char`	`SIGNIFICANCE n`
`TRAILING SIGN string`	`UPSHIFT`
`VALUES value-set`	

The remaining options are as follows:

**[DEFAULT] [ITEM] DATATYPE type [SIGNED|UNSIGNED]
[SIZE n [BYTES]] [OCCURS n]**

Defines the default item datatype and size for element definitions created from this usage. DEFAULT and ITEM are for documentation only. The datatype specified in the usage definition can be changed in the element definition, or in any particular item definition. For more information, see the ELEMENT statement on (p. 43) and the ITEM statement on (p. 74).

DATATYPE type

Specifies how item values are stored. The PowerHouse datatypes are:

CHARACTER	DATETIME	FLOAT
FREEFORM	G_FLOAT (**OpenVMS**)	INTEGER
INTERVAL	JDATE	PACKED
PHDATE	VARCHAR	VMSDATE (**OpenVMS**)
ZDATE	ZONED	

SIGNED|UNSIGNED

Indicates whether the datatypes INTEGER, PACKED and ZONED are SIGNED or UNSIGNED. INTEGER SIGNED can store negative values; INTEGER UNSIGNED cannot. ZONED and PACKED datatypes can store positive or negative numbers, regardless of whether SIGNED or UNSIGNED is specified. The SIGNED and UNSIGNED options are mutually exclusive and must immediately follow the datatype.

Limit: Applies only to INTEGER, PACKED, and ZONED. SIGNED and UNSIGNED must immediately follow the datatype.

Default: SIGNED for datatypes INTEGER and PACKED; UNSIGNED for ZONED.

SIZE n [BYTES]

Specifies the storage size, in bytes, of items based on this element.

DESCRIPTION string [[,]string]...

This is the only attribute of a usage that is not used to set an element attribute. This description serves as documentation for the usage itself. The description can be seen in QSHOW reports.

Limit: 60 characters per string.

Discussion

The USAGE statement defines usages or references existing usages. Usages are used in the ELEMENT statement to set element attributes to predetermined values. If an element has a usage, most of its attributes are taken from the usage definition rather than from the element definition. A single usage can serve as a template for many element definitions.

Standard usages are preloaded in all PDL dictionaries with the PRELOADED option of the CREATE DICTIONARY statement. For information about standard dictionary usages, see Chapter 3, "Standard Elements and Usages".

Example

The attributes assigned to the element, TRANSNO, could be the same for a number of other elements in this dictionary. Instead of defining the same attributes for a number of different elements, you could create a usage describing an element of this type

```
>   USAGE IS INVENID &
>       NUMERIC SIZE 7 &
>       PICTURE "^^^-^^^^" &
>       SIGNIFICANCE 8
```

Once the usage is defined, the ELEMENT statement for TRANSNO is

```
>   ELEMENT TRANSNO &
>       USAGE INVENID
```

USE

Processes PDL statements contained in source statement files.

Syntax

USE filespec [DETAIL|NODETAIL] [LIST|NOLIST]

filespec

Names the file containing PDL source statements.

OpenVMS, UNIX, Windows: The default extension is .pdl.

DETAIL|NODETAIL

DETAIL writes the contents of the source file to PDL's temporary save file. NODETAIL doesn't write the contents of the source file to the temporary save file; it only writes the USE statement to the temporary save file.

Default: DETAIL

LIST|NOLIST

LIST displays the statements as they are read; NOLIST does not.

Default: LIST

Discussion

The USE statement instructs PDL to read the named file for statement input. PDL reads and interprets each statement as if it had been entered from the terminal. The file can contain other USE statements.

For more information on locating files, see Chapter 1, "Running PowerHouse", in the *PowerHouse Rules* book.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the USE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Example

This USE statement processes a PDL source file without listing it. It copies the statements in UXDICTS but doesn't copy the statement, USE UXDICTS, to PDL's temporary save file.

```
> USE UXDICTS DETAIL NOLIST
```

USER MODE (OpenVMS)

Defines a user mode and assigns its members.

Syntax

USER MODE name [option]...

name

Names a new user mode, or one that has already been defined.

Limit: 64 characters. Must begin with a letter. The rest of the name can contain letters, digits, and special name characters that are defined in the dictionary.

Options

The options are LOGONID, PASSWORD, PORTID, and UIC.

LOGONID logonid [[,] logonid]...

Assigns user names that are potential members of this user mode. The logonid is an OpenVMS user name.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is LOGONID, and if the user's logon ID is one of the logonids, the user becomes a member of the user mode.

Limit: 12 characters. Logonids must be enclosed in quotation marks if they begin with a numeric.

PASSWORD password [[,] password]...

Assigns passwords that users must enter to become a member of this user mode. A password is a 31-character string, with or without quotation marks.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is PASSWORD, PowerHouse prompts the user for a password. If the user enters one of the passwords listed after the PASSWORD option of a USER MODE statement, the user becomes a member of that user mode.

Limit: 31 characters.

PORTID portid [[,] portid]...

Assigns port IDs. A portid is an OpenVMS device name. A user who accesses PowerHouse through one of the port IDs listed becomes a member of the user mode.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is PORTID, and if the user's port ID is included as one of the portids, the user becomes a member of the user mode.

Limit: 10 characters

UIC [gid,mid] [[,] [gid,mid]]...

Lists members of the user mode by their OpenVMS numeric UICs. The UIC is defined by a gid (group id number) and a mid (member id number). The gid can be from 0-37776 (octal) and the mid can be from 0-177776 (octal). The gid and mid must be declared as a pair. The brackets around [gid,mid] are required syntax.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is UIC, and if the user's UIC is one of the UICs listed, the user becomes a member of the user mode.

WEBLOGONID name|string [[,] name|string]...

Assigns user names that are potential members of this user mode.

The name option can be used if the username is a valid PowerHouse name, otherwise, string must be used. Depending on the platform where they originate, user names are case sensitive.

When a user opens a dictionary from any PowerHouse component, PowerHouse checks the ID method in the APPLICATION SECURITY ID METHOD option of the SYSTEM OPTIONS statement. If the ID method is WEBLOGONID, and if the user's WEBLOGONID is one of the web logonids listed, the user becomes a member of the user mode.

Discussion

A user mode defines a set that have similar characteristics. The EXECUTION TIME PARAMETERS statement assigns default data file locations to user modes.

Users who aren't members of a specific user mode have no execution time parameters. For more information, see the EXECUTION TIME PARAMETERS statement on (p. 57).

Security and Compiled Applications

The following information is compiled into QUIZ reports, QTP runs, and QUICK screens:

- ASC ID Method
- the name of the ASC and the association of the ASC with records and elements
- User Modes and locations specified in the ETP statement

If any of this information changes, the corresponding screens, reports, and runs need to be recompiled.

The membership of ASCs and User Modes is dynamic. You can add IDs to and delete IDs from ASCs without requiring that screens, reports, and runs be recompiled.

Examples

For examples of the USER MODE statement, see the examples for the EXECUTION TIME PARAMETERS statement on (p. 57).

Chapter 3: Standard Elements and Usages

Overview

All dictionaries built with PDL, PHDPDL, and PHDMAINT can be preloaded with a set of standard usages. The following tables describe these standard elements and usages.

For information about how to create your own usages, see (p. 118).

	FACTOR	**FILLER**	**FILLER-NUMERIC**
Type/Size	Numeric/9	Character/1	Numeric/1
Item Datatype	FLOAT Size 8	CHARACTER Size 1	INTEGER SIGNED Size 2
BWZ (Blank When Zero)	Yes	n/a	No
Decimal Positions	5	n/a	0
Leading Sign	"-"	n/a	" "
Pattern	"#*(.#<#<#<#<#<)<"	n/a	n/a
Input Scale	0	n/a	0
Output Scale	5	n/a	0
Picture	" ^^^^.^^^^^"	n/a	"^"
Significance	7	n/a	1
Shift Input	n/a	n/a	n/a
Trailing Sign	n/a	n/a	n/a
Description	For elements with fractional values intended for use in calculations. All values are stored as floating point numbers.	A standard element added to a record-structure to make it longer. It can be used once in any record-structure or item substructure to specify an area of storage that is unused. Because FILLER is a character element, its data area is initialized to spaces. You can define other elements, such as FILLER-1, FILLER-2, etc., if you need more than one.	The same characteristics as FILLER. Because FILLER-NUMERIC is a numeric element, its data area is initialized to zeros.
Examples	UNIT_PRICE		

	FLAG	**ID**	**INTERVAL**
Type/Size	Character/1	Character/0	Numeric/0
Item Datatype	CHARACTER Size 1	CHARACTER Size 0	INTERVAL Size 8
BWZ (Blank When Zero)	n/a	n/a	Yes
Decimal Positions	n/a	n/a	0
Leading Sign	n/a	n/a	" "
Pattern	n/a	n/a	n/a
Input Scale	n/a	n/a	n/a
Output Scale	n/a	n/a	9
Picture	n/a	n/a	"^^^^^ ^^.^^.^^.^^^"
Significance	n/a	n/a	15
Shift Input	Upshift	Upshift	n/a
Trailing Sign	n/a	n/a	n/a
Description	For elements that serve as checks, flags, or selection criteria. FLAG has an initial value set to "N". The value can be changed to "Y".	For character elements to be used in an item definition for which an index is declared.	For elements used as amounts of time.
Examples	BILINGUAL in a personnel application. Each time a record is created, the corresponding item BILINGUAL is assigned an initial value of "N" (which can be changed to "Y" if the employee is bilingual).	DEPARTMENTCODE represents codes for different departments in a purchase order tracking application.	TIMETOPAY in a purchase order system.

	MONEY	**MONEY-CR**	**MONEY-DR**
Type/Size	Numeric/9	Numeric/9	Numeric/9
Item Datatype	INTEGER SIGNED Size 4	INTEGER SIGNED Size 4	INTEGER SIGNED Size 4
BWZ (Blank When Zero)	No	n/a	n/a

	MONEY	**MONEY-CR**	**MONEY-DR**
Decimal Positions	2	2	2
Leading Sign	n/a	n/a	n/a
Pattern	n/a	n/a	n/a
Input Scale	2	2	2
Output Scale	0	0	0
Picture	"^,^^^,^^^.^^ "	"^,^^^,^^^.^^ "	"^,^^^,^^^.^^DR"
Significance	5	6	6
Shift Input	n/a	n/a	n/a
Trailing Sign	"-"	"CR"	"CR"
Description	For currency elements that don't require specialized leading or trailing signs to display negative numbers.	For currency elements for which negative values are displayed with a CR notation to indicate a credit.	For currency elements for which negative values are displayed with the CR notation to indicate a credit, and positive values are displayed with the DR notation to indicate a debit.
Examples	SALARY in a payroll application.	BALANCEOWING in a customer invoicing application.	DEFERREDTAXES in a financial statement reporting application.

	MONEY-PR	**NAME**	**NUMERIC-ID**
Type/Size	Numeric/9	Character/20	Numeric/0
Item Datatype	INTEGER SIGNED Size 4	CHARACTER Size 20	ZONED Unsigned Size 0
BWZ (Blank When Zero)	No	n/a	Yes
Decimal Positions	2	n/a	0
Leading Sign	"("	n/a	n/a
Pattern	n/a	n/a	n/a
Input Scale	2	n/a	0
Output Scale	0	n/a	0
Picture	"^,^^^,^^^.^^"	n/a	"^^^^^^^^^^^^^^^^^^^^^^^^^^^^"
Significance	5	n/a	28
Shift Input	n/a	n/a	n/a

	MONEY-PR	NAME	NUMERIC-ID
Trailing Sign	")"	n/a	n/a
Description	For currency elements for which negative values are displayed in parentheses.	For standard elements that aren't likely to be used in an item definition for which an index is declared.	For numeric elements that are likely to be used in an item definition for which an index is declared.
Examples	NETPROFIT in a financial reporting application.	LASTNAME in a personnel application.	EMPLOYEENUMBER for which an index is declared in a personnel record-structure.

	PERCENT	PHONE	POSTAL-CD
Type/Size	Numeric/5	Numeric/10	Character/6
Item Defaults	FLOAT Size 4	INTEGER UNSIGNED Size 8	CHARACTER Size 6
BWZ (Blank When Zero)	Yes	Yes	n/a
Decimal Positions	2	0	n/a
Leading Sign	"-"	" "	n/a
Pattern	n/a	n/a	"^#^#^#"
Input Scale	-2	0	n/a
Output Scale	4	0	n/a
Picture	"^^^.^^%"	"^^^ ^^^-^^^^"	"^^^ ^^^"
Significance	5	8	n/a
Shift Input	n/a	n/a	Upshift
Trailing Sign	n/a	n/a	n/a
Description	For elements with fractional values used in calculations and displayed as percentages.	For elements that represent North American telephone numbers. The values are displayed as 555-1212 or 613 555-1212.	For elements that represent Canadian postal codes.
Examples	CUSTOMERDISCOUNT in an invoicing application. A value of 15 is stored as .155 and displayed as 150%.	HOMEPHONE in a personnel system.	POSTALCODE in a mailing list application.

	QUANTITY	**TIME**	**TIME-STAMP**
Type/Size	Numeric/0	Numeric/4	Numeric/8
Item Datatype	INTEGER SIGNED Size 2	INTEGER UNSIGNED Size 2	INTEGER UNSIGNED Size 4
BWZ (Blank When Zero)	Yes	n/a	Yes
Decimal Positions	0	0	0
Leading Sign	"-"	n/a	n/a
Pattern	n/a	"((1<#|2(0|1|2|3))<(0|1|2 |3|4|5))<#"	n/a
Input Scale	0	0	0
Output Scale	0	0	0
Picture	"^,^^^,^^^,^^^,^^^,^^ ^"	"^^:^^"	"^^:^^:^^.^^."
Significance	1	5	11
Shift Input	n/a	n/a	n/a
Trailing Sign	n/a	n/a	n/a
Description	For numeric elements that are not likely to be used in an item definition for which an index is declared and for items for which the values will be integers.	For elements that represent time values in hours and minutes.	For elements that represent precise times that include hours, minutes, seconds, and hundredths of seconds. Generally, items with this usage are initialized with the current system time.
Examples	ONHAND, ONORDER, or ONREQUEST in an inventory control application.	**MPE/iX:** SHIFTSTARTTIME in a payroll application. **OpenVMS, UNIX, Windows:** HOURSWORKED in a payroll application.	

	ZIP-CODE
Type/Size	Numeric/5
Item Datatype	INTEGER UNSIGNED Size 4
BWZ (Blank When Zero)	Yes
Decimal Positions	0

	ZIP-CODE
Leading Sign	n/a
Pattern	n/a
Input Scale	0
Output Scale	0
Picture	`"^^^^^"`
Significance	5
Shift Input	n/a
Trailing Sign	n/a
Description	For elements that represent American zip codes.

Chapter 4: QSHOW Statements

Overview

QSHOW lets you display or print the names and attributes of PowerHouse

- files and databases
- record-structures
- elements
- indexes
- usages

QSHOW uses the syntax and language rules discussed in Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Summary of QSHOW Statements

The following table summarizes the purpose of each QSHOW statement:

Statement	Purpose
EXIT	Ends a QSHOW session.
GENERATE	Generates PDL or SQL source statement definitions from a compiled dictionary.
QUIT	Ends a QSHOW session.
REVISE	Invokes the editor to edit the current temporary save file or a specified file.
SAVE	Saves QSHOW source statements in a file.
SET	Changes the default settings for a session.
SHOW	Reports dictionary definitions.
USE	Processes QSHOW source statements contained in a file.

EXIT

Ends a QSHOW session.

Syntax

EXIT

Discussion

The EXIT statement ends the QSHOW session and returns control to the operating system or to the invoking program.

EXIT and QUIT are interchangeable.

GENERATE

Generates PDL or SQL source statement definitions from a compiled dictionary.

Syntax

With SET LANGUAGE PDL (default):

GENERATE ALL

**GENERATE {DATABASE|ELEMENT|FILE|TRANSACTION|
USAGE} [name-option|ALL]**

GENERATE SYSTEMS

With SET LANGUAGE SQL:

GENERATE DATABASE [name-option|ALL]

Options

ALL

Generates PDL definitions for all the entities in the data dictionary. The GENERATE ALL statement generates all the types of PDL statements that the GENERATE FILE statement generates. In addition, the GENERATE ALL statement generates

- the USAGE and ELEMENT statements for usages and elements that are in the data dictionary but are not used in any item in any record-structure
- the ASC and PERMIT statements for application security classes; the ETP and USER MODE statements (**OpenVMS**); the DSC statements (**PHD Dictionaries, OpenVMS**).

  These statements are generated only if the SET SECURITY statement is used before using the GENERATE ALL statement and the dictionary creator and the user are the same person

  or

  the user has Account Manager capabilities for the account where the dictionary is located or the user has System Manager capabilities. (**MPE/iX**)

  or

  the user is a member of the appropriate dictionary security class. (**PHD Dictionaries, OpenVMS**)

  General users cannot generate any security information. Application managers can generate ASC, USER MODE, and PERMIT statements but not DSC statements. Dictionary managers and creators can generate all security information.
- the SYSTEM OPTIONS statements as specified in the data dictionary
- DATABASE statements
- TRANSACTION statements

DATABASE

With SET LANGUAGE PDL in effect (default), generates PDL DATABASE statements for the database or databases specified by the name-option.

With SET LANGUAGE SQL in effect, generates the complete SQL from the relational database specified by the name-option. If the name-option is a non-relational file, QSHOW will generate the SQL based on the dictionary definition. The generated SQL is created in a file named qshogen.sql (UNIX, Windows), QSHOGEN.TXT (OpenVMS), or QSHOSQL (MPE/iX). QSHOW does not provide the ability to change the name of the output file.

FILE

Generates the PDL statements for entities in the file(s) specified by the name-option. This statement produces the following types of PDL statements:

- a USAGE statement for each usage that is used by elements referenced by the specified files
- an ELEMENT statement for each element that is used as an item in the specified files
- a FILE statement for each specified file
- a RECORD statement for each record-structure that is used in the specified files
- an ITEM statement for each item that is used in the specified files
- a BEGIN STRUCTURE statement and an END STRUCTURE statement for each substructure that is used in the specified files
- an INDEX and SEGMENT statement for each index and segment that is used in the specified files.

ELEMENT

Generates PDL statements for elements and associated usages specified by the name-option. This option produces the following types of PDL statements:

- a USAGE statement for each usage that's referenced by a specified element
- an ELEMENT statement for each specified element

TRANSACTION

Generates PDL TRANSACTION statements.

SYSTEMS

Generates the SYSTEM OPTIONS statements specified in the data dictionary.

USAGE

Generates PDL code for a set of usages specified by the name-option.

Name-Options

The name-options are ALL, namelist, PATTERN, and SOUNDEX.

ALL

Specifies that definitions should be generated for all files, elements, or usages defined in the dictionary.

nameset

Identifies files, elements or usages by a combination of single names, lists of names, and ranges of names, using the general form:

name [TO name] [,name [TO name]]...

The name list is restricted to ten names or less.

PATTERN string

Identifies files, elements, or usages whose names match the pattern specified by the string. For more information, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

SOUNDEX (string[,n])

Identifies files, elements, or usages whose names sound like the string. For example, the SOUNDEX string "stockfile" retrieves names such as STOCKFL and STOCK-FL. For more information, see the section, "Using the SOUNDEX Option", in Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Discussion

When SET LANGUAGE PDL is in effect, the GENERATE statement generates PDL code from the definitions in a PowerHouse data dictionary.

When SET LANGUAGE SQL is in effect, the GENERATE DATABASE statement generates SQL from a database or from dictionary files. The latter allows you to generate skeleton SQL from non-relational files. To convert multiple non-relational files into a single relational database, generate the SQL for each file one by one. After each file, copy the results from qshogen.sql into another text file and make the appropriate changes.

How It Works

The GENERATE statement writes generated PDL or SQL statements to the file, qshogen. Within a given QSHOW session, the first time you generate PDL or SQL statements and qshogen exists, you are asked whether you want to overwrite the file (**MPE/iX, UNIX, Windows**) or to create a new version (**OpenVMS**). Subsequent GENERATE statements cause QSHOW to append statements to the qshogen file. If you explicitly specify the name of a file in which to store generated PDL statements, QSHOW will append statements to that file.

OpenVMS, UNIX, Windows: The default extension for a source PDL file is .pdl. The fixed extension for a source SQL file is .sql.

The file to which PDL statements are written can be changed using the statement

```
> SET GENERATE DEVICE options
```

The file to which SQL statements are written can only be qshogen.sql (UNIX, Windows), QSHOGEN.TXT (OpenVMS), or QSHOSQL (MPE/iX).

For more information, see the SET statement (p. 141).

Option Defaults

QSHOW only generates PDL statements for entity attributes explicitly set by the PDL statements used to create the dictionary. It doesn't generate PDL statements for entity attributes that have default values. If you load generated PDL statements in a dictionary with different system options, the resulting dictionary may contain different attributes.

Modifying Existing Dictionaries

You can't change existing ELEMENT, USAGE, FILE, or RECORD definitions in PDL or PHDPDL. If you have to change one of these, you must generate the full PDL code for the existing dictionary with the GENERATE statement in QSHOW. Then change the definition(s), create the dictionary, use the PDL source statements, and load the new dictionary as follows:

1. Use GENERATE in QSHOW to produce a source file that contains the PDL statements needed to recreate the dictionary.
2. Use the system editor to modify the generated source code.
3. Use CREATE DICTIONARY, USE, and LOAD to recreate and load the dictionary.

OpenVMS: Alternatively, you can use the PhD Screen System to make the needed changes to PHD dictionaries.

Loading a New Dictionary with Generated Definitions

The GENERATE statement loads a new dictionary with definitions from another dictionary, which eliminates the need to re-enter definitions. The following example shows the statements entered, using QSHOW and PDL or PHDPDL, to load a new dictionary with generated definitions.

In QSHOW

```
> SET DICTIONARY OLD
> SET LANGUAGE PDL
> GENERATE ALL
> EXIT
```

This creates a source file named qshogen.pdl (**OpenVMS, UNIX, Windows**) or qshogen (**MPE/iX**). In PDL or PHDPDL, enter

```
> CREATE DICTIONARY NEW NOT PRELOADED
> USE QSHOGEN
> LOAD
> EXIT
```

The SET DICTIONARY statement points QSHOW to the OLD dictionary. The SET LANGUAGE statement tells QSHOW to generate the definitions in PDL. The GENERATE ALL statement produces definitions for all the entities in the OLD dictionary.

In PDL or PHDPDL, the dictionary NEW is created using the CREATE DICTIONARY statement. The USE statement processes the definitions in the designated file, qshogen. The LOAD statement enters the definitions into the NEW dictionary.

It is important to remember to use the NOT PRELOADED option of the CREATE DICTIONARY statement if loading definitions from an old dictionary that does not use the standard usages. If source code is generated for the entire dictionary, the usages from the old dictionary are loaded into the new one.

QUIT

Ends a QSHOW session.

Syntax

```
QUIT
```

Discussion

The QUIT statement ends your QSHOW session and returns control to the operating system or to the invoking program.

QUIT and EXIT are interchangeable.

REVISE

Edits the current temporary save file or a specified file.

Syntax

REVISE [*|filespec [option]...]

*

An asterisk (*) tells QSHOW to edit the current qshosave file. The qshosave file is a temporary file that QSHOW opens at the beginning of a session.

All QSHOW statements except the save and revise statements are recorded in the qshosave file as you enter them. The asterisk is not required if you are editing the qshosave file without changing any default options. However, the asterisk is required if you are overriding any of the default options, as in

```
> revise * nouse
```

filespec

The filespec is the name of an existing permanent file. If this file does not contain QSHOW statements, use the NOUSE option so that QSHOW does not try to process the file when you exit from the system editor.

Options

The options are DETAIL, NODETAIL, LIST, NOLIST, USE, and NOUSE.

DETAIL|NODETAIL

DETAIL copies the contents of the revised file into the qshosave file when you exit from the system editor; NODETAIL does not. If you are revising a permanent file with the use statement and the NODETAIL option in effect, then a use statement is written to the current qshosave file and is invoked when you exit from the editor, as in

```
> USE ORDERS NODETAIL
```

Limit: The NODETAIL option is not valid when you are revising qshosave.

Default: DETAIL

LIST|NOLIST

LIST displays each statement from the revised file as it is executed; NOLIST does not.

Default: LIST

USE|NOUSE

USE processes the revised statements when you exit from the system editor. NOUSE returns you to QSHOW at the point from which you left it without processing the revised statements.

Default: USE

Discussion

The REVISE statement lets you use the system editor to edit either the qshosave file or a permanent file from within QSHOW. The qshosave file is edited by default. When you enter the REVISE statement without a filename, a CANCEL CLEAR statement is automatically applied; when you enter the REVISE statement with a filename, the automatic CANCEL CLEAR statement is not in effect.

MPE/iX:	By default the REVISE statement chooses EDITOR/3000 as the system editor. You can change this by using a file equation for COGEDITR.

UNIX, Windows:	By default, the REVISE statement uses the editor defined by the environment variable, PHEDIT. If PHEDIT is not defined, the system checks the environment variable, EDITOR. If you have not defined either of these variables, the REVISE statement fails.
OpenVMS:	By default, the REVISE statement chooses EDT as the system editor. You can change this by using the DCL symbol, PHEDIT.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the REVISE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

SAVE

Saves QSHOW source statements in a file.

Syntax

SAVE filespec [CLEAR]

filespec

Names a file that will contain the QSHOW statements.

MPE/iX, UNIX, Windows:	If QSHOW finds an existing file with the same name, it prompts for confirmation before overwriting the existing file (unless SET NOVERIFY DELETE is in effect).
OpenVMS:	If QSHOW finds an existing file with the same name, it prompts for confirmation before creating a new version.

CLEAR

Removes any source statements in the temporary save file, qshosave, once the contents are copied to a permanent file.

Discussion

The SAVE statement relates to QSHOW's temporary source statement save file, qshosave (**MPE/iX**) or qshosave.txt (**OpenVMS, UNIX, Windows**). Statements are written to this file as you enter them. The SAVE statement itself is not included in the file.

The SAVE statement creates a permanent copy of qshosave at the point where the SAVE statement was entered. You can use the saved contents as a source file for documentation and future changes, or as a working file for modification using the system editor. The saved statements can also be processed by QSHOW with the USE statement.

The CLEAR option clears the temporary save file after its contents have been saved so that you can enter and then save a new set of QSHOW statements in the same session. To clear the temporary save file without saving its contents, use the SET SAVE CLEAR statement.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the REVISE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

SET

Changes the default settings for a session.

Syntax

SET DEFAULT|option...

DEFAULT

This resets all QSHOW SET options to the following default values:

CLOSE	PAGE LENGTH 23 (for terminals), 60 (for printers)
COPIES 1	PRIORITY 8 (**MPE/iX**)
LANGUAGE PDL	QUEUE SYS$PRINT (**OpenVMS**)
NOCOMPRESSED	REPORT DEVICE TERMINAL
NOPASSWORD (**MPE/iX**)	SORT (**MPE/iX**)
NOPRINT	XREF
NOSECURITY	

SET DEFAULT doesn't reset the SET DICTIONARY option. If DEFAULT is used, it must be the only option of the SET statement.

SET statements override default options normally in effect during a QSHOW session.

Options

SET Options		
CLOSE|NOCLOSE	COMPRESSED|NOCOMPRESSED	COPIES
DICTIONARY	DOWNSHIFT|UPSHIFT|NOSHIFT	GENERATE DEVICE
LANGUAGE	PAGE LENGTH	PASSWORD|NOPASSWORD
PRINT|NOPRINT	PRIORITY	REPORT DESTINATION
REPORT DEVICE	SAVE CLEAR	SECURITY|NOSECURITY
SORT|NOSORT	XREF|NOXREF	

CLOSE|NOCLOSE

The CLOSE option closes the output file after each QSHOW report is sent to the system printer. Reports are not printed until the output file is closed. The NOCLOSE option leaves the output file open until a SET DEVICE TERMINAL or SET CLOSE statement is entered, or until the QSHOW session is ended.

Default: CLOSE

COMPRESSED|NOCOMPRESSED

COMPRESSED produces QSHOW reports that use the smallest possible format by eliminating headings, page numbers, page breaks, and white space. NOCOMPRESSED restores the reports to the default format.

Default: NOCOMPRESSED

COPIES n

Sets the number of copies to be produced when routing output to a printer.

Default: 1

DICTIONARY filespec [TYPE PHD|PDC]

Names the dictionary for the current QSHOW session. All subsequent QSHOW statements report on entities in this dictionary.

The SET DICTIONARY statement can be used any number of times in a single session. SET DICTIONARY overrides the **dictionary** program parameter.

[TYPE PHD|PDC] (OpenVMS)

Specifies the default dictionary type. If the TYPE option is specified in a PowerHouse component, it applies to subsequent SET DICTIONARY statements in the component.

When searching for a dictionary, PowerHouse limits searches to the dictionary type specified by the TYPE option. If the TYPE option is not specified, PowerHouse searches first for a PHD dictionary, then a PDC dictionary.

Default: PHD

DOWNSHIFT|UPSHIFT|NOSHIFT

Specifies that the values of entered identifiers be shifted to lowercase, uppercase, or left as entered.

Default: UPSHIFT

GENERATE DEVICE option (OpenVMS)

Directs the generated output of QSHOW to the specified location. The options included are:

TERMINAL|[PRINTER [printer-option]]|[DISC [NAME filespec]]

Default: The default output device is DISC. If output is directed to disk and no filename is specified, the default output file is qshogen.pdl (when used after SET LANGUAGE PDL) or qshogen.sql (when used after SET LANGUAGE SQL).

For information on the printer options, see (p. 145).

LANGUAGE COBOL|PDL|SQL

COBOL (MPE/iX)

The COBOL option is used with the QSHOW GENERATE statement to generate COBOL from a PDL dictionary. For example, to generate COBOL definitions for all the files in your dictionary, enter

```
> SET LANGUAGE COBOL
> GENERATE FILE ALL
```

For more information about generating COBOL, see Chapter 7, "QCOBLIB Utility".

PDL

Specifies that PowerHouse Definition Language (PDL) is to be generated by the GENERATE statement.

The LANGUAGE PDL option provides compatibility with other versions of PowerHouse.

SQL

Specifies that SQL statements are to be generated by the GENERATE statement.

PAGE LENGTH n

Specifies the number of lines in a QSHOW report page.

Default: 23 for terminals; 60 for printers.

PASSWORD|NOPASSWORD (MPE/iX)

PASSWORD reports IMAGE database passwords, and KSAM and MPE lockwords, in file reports; NOPASSWORD does not.

Limit: SET PASSWORD has no effect unless you are the dictionary creator, or you have system manager or account manager capability.

Default: NOPASSWORD

PRINT|NOPRINT

MPE/iX:	PRINT causes the statements you have entered during the QSHOW session to be sent to a spool file (STDPRINT). When you end the session or issue a SET NOPRINT statement, the spool file is closed and the output is printed on the system printer. NOPRINT immediately closes a previously opened spool file, and prints the output.
OpenVMS, UNIX, Windows:	PRINT sends the source listing to the default printer; NOPRINT does not.

Default: NOPRINT

PRIORITY n (MPE/iX)

Sets the MPE/iX priority for printer output.

Limit: The priority can be from 0 to 13.

Default: 8

For information about the PRIORITY printer option on **OpenVMS**, see (p. 147).

[REPORT] DESTINATION printername (UNIX, Windows)

Specifies the printer where the report is to be sent.

Use the DESTINATION option instead of the PRINTER string option to identify a specific printer if operating system access is not allowed.

The DESTINATION option is ignored if a string is specified on the SET DEVICE PRINTER statement.

Limit: Valid for SET DEVICE PRINTER only.

Default: Determined by the system used.

[REPORT] DEVICE [DISC|PRINTER|TERMINAL]

Specifies the destination output device for the QSHOW report. DEVICE also closes the output file, and sets the default page length.

[DISC [NAME filespec]]

Specifies that the output is directed to disc.

Defaults: If you don't enter a name when you select the DISC option, the default file name is qsholist (**MPE/iX**) or qsholist.txt (**OpenVMS, UNIX, Windows**).

PRINTER (MPE/iX)
PRINTER [printer-option] (OpenVMS)
PRINTER [string] (UNIX, Windows)

Specifies that the output is directed to the printer.

string (UNIX)

The optional string is the UNIX Shell command used to send the report to the printer.

A string is not valid if the **noaccess** program parameter is specified or the OSACCESS resource file option equals OFF. A syntax error will result.

If the string is omitted, PowerHouse follows these steps to determine the appropriate values:

1. If a string was used in the last SET DEVICE PRINTER [string] statement, PowerHouse uses that value. The DESTINATION option is ignored.
2. If the value of the environment variable PH_PRINTER is defined, PowerHouse uses this value. The DESTINATION option is ignored. QSHOW assumes this argument is available in the PH_PRINTER environment variable.
3. The system default string is used. If specified, PowerHouse uses the DESTINATION, option; otherwise the default DESTINATION value is used.

   If the string is null, options 2 and 3 are used to obtain a value.

For a list of the **OpenVMS** printer options for the REPORT DEVICE PRINTER statement, see (p. 145).

Default: For batch jobs, the default is PRINTER (**OpenVMS**).

TERMINAL

Specifies that the output is directed to the terminal.

Defaults:

MPE/iX, UNIX, Windows:	TERMINAL for interactive sessions. The statement is ignored for batch jobs.
OpenVMS:	TERMINAL for interactive sessions. For batch jobs, the TERMINAL option is not valid, and the default is PRINTER.

SAVE CLEAR

Immediately removes any source statements from the temporary save file. The temporary save file is qshosave (**MPE/iX**) or qshosave.txt (**OpenVMS, UNIX, Windows**).

For more information, see the SAVE statement on (p. 140).

A CLEAR is always performed after a SAVE.

SECURITY|NOSECURITY

SECURITY reports application security in file and element reports. It enables the reporting of security specifications and the generation of security specifications by the PDL code generator.

NOSECURITY does not report application security in file and element reports. SET SECURITY has no effect unless you are the creator of the dictionary.

The following table provides information about security for PDC and PHD dictionaries on different platforms:

	PDC Type Dictionaries	PHD Type Dictionaries
OpenVMS:	SHOW APPLICATION SECURITY and SHOW USERS statements are only available to the dictionary creator.	When SET SECURITY is entered, QSHOW checks the access to the dictionary to identify the dictionary user. If the dictionary security type PASSWORD is specified, QSHOW prompts the user for a password. SHOW APPLICATION SECURITY and SHOW USERS statements are only available to dictionary users who have application manager or dictionary manager access. SHOW DICTIONARY SECURITY statement is only available to dictionary users who have dictionary manager access.
UNIX, Windows:	SHOW APPLICATION SECURITY and SHOW USERS statements are only available to the dictionary creator.	
MPE/iX:	SHOW APPLICATION SECURITY and SHOW USERS statements are only available to the dictionary owner or to those who have account manager (AM) privileges for the account in which the dictionary resides or have system manager (SM) privileges.	

Default: NOSECURITY

SORT|NOSORT (MPE/iX)

SORT produces alphabetically sorted lists when reporting files, elements, records, and usages. NOSORT suppresses the sorting of PowerHouse entities. Choosing NOSORT can save time when reporting entities from large dictionaries.

Default: SORT

XREF|NOXREF

XREF reports cross-referenced information in each report. For example, a cross-referenced element in a QSHOW report contains a list of records in which the element name appears. NOXREF doesn't report cross-referenced information. You can save time when reporting entities from large dictionaries by choosing not to report cross-referenced information.

Default: XREF

DEVICE PRINTER Options (OpenVMS)

SET REPORT DEVICE PRINTER Options		
AFTER	BURST|NOBURST	COPIES
FLAG|NOFLAG	FORMS	HOLD|NOHOLD

SET REPORT DEVICE PRINTER Options

IDENTIFY|NOIDENTIFY	LOWERCASE|NOLOWERCASE	NOCHARACTERISTIC|CHARACTERISTIC
NOTE	NOTIFY|NONOTIFY	OPERATOR|NOOPERATOR
PRIORITY	QUEUE	RESTART|NORESTART
TRAILER|NOTRAILER		

AFTER[absolute-time]+|- delta time]

Specifies the time at which the report is to start executing. An absolute-time is a specific date and/or time of day. It has the general form:

**{dd-mmm[-yyyy][:[hours][:[minutes][:[seconds]
[.[hundredths]]]]]} | {TODAY|TOMORROW|YESTERDAY}**

A delta-time is an offset from the current time to a time in the future. It has the general form:

[days][-[hours]][:[minutes][:[seconds][.[hundredths]]]]

You must indicate either an absolute time, a delta time, a combination of both absolute and delta time or the time indicators TODAY, TOMORROW, and YESTERDAY.

In the following absolute time example, the report executes at noon on May 15, 2001:

```
> SET REPORT DEVICE PRINTER AFTER 15-MAY-2001:12
```

In the next example, the report executes according to a delta time, three hours from the current time:

```
> SET REPORT DEVICE PRINTER AFTER 15-MAY-2001:12
```

In the next example, the report executes according to a delta time, three hours from the current time:

```
>  SET REPORT DEVICE PRINTER AFTER +3
```

In this final example, the report executes at 11:00 p.m. on the current date:

```
> SET REPORT DEVICE PRINTER AFTER TOMORROW -1
```

Default: The default is the time that the job reaches the top of the queue.

BURST|NOBURST

BURST specifies that a burst page is to be printed prior to the flag page at the front of the report. This enables you to easily identify where one report ends and another begins. The burst page contains the same information as the flag page, but prints over the perforation. When BURST is specified, FLAG need not be specified. NOBURST does not print the burst page.

Default: BURST

COPIES n

Specifies the number (n) of copies of a report to be printed.

Limit: Valid only for the SET REPORT DEVICE PRINTER statement. The maximum number of copies is 255.

Default: 1

FLAG|NOFLAG

FLAG indicates that a flag page is to be printed at the front of the report. The flag page contains information about the file being printed. This option need not be specified if the BURST option has been used, as a flag page is automatically produced when BURST is specified. NOFLAG suppresses the flag page.

Default: FLAG

FORMS number|string

Specifies which form is to be used for the report. The value entered can be a number or a string of characters.

A string is a series of displayable characters (letters, numbers, or special characters) in double or single quotation marks. This string can be an actual form name or a logical name for a form. If FORM is not specified, the system's regular stock forms are used. To see the forms that are available for your system, use the DCL command $SHOW QUEUE/FORM.

Limit: If an actual form name is used, the string can be up to 31 characters in length. If a logical name is used, the string can be up to 255 characters in length.

HOLD|NOHOLD

HOLD indicates that the job is to be held in queue until specifically released; NOHOLD does not.

Default: NOHOLD

IDENTIFY|NOIDENTIFY

IDENTIFY indicates that the message containing the job number and queue message is to be displayed when the report is sent to the print queue; NOIDENTIFY does not.

Default: IDENTIFY

LOWERCASE|NOLOWERCASE

LOWERCASE indicates that the report is to be printed on a printer that supports both uppercase and lowercase letters. NOLOWERCASE indicates that the report can be printed on a printer that supports either uppercase or lowercase letters.

Default: LOWERCASE

NOCHARACTERISTIC|CHARACTERISTICS number|string [,number|string]...

CHARACTERISTICS specifies one or more characteristics that you can use in defining the printing format of the report.

The DCL command, $SHOW QUEUE|CHAR, shows you the characteristics that are in effect for your system. NOCHARACTERISTIC specifies that any previously set characteristics are to be canceled.

Limit: A maximum of 127 characteristics can be defined. If numbers are specified, they must be within the range of 0 to 127. If you indicate a string, you can specify a maximum of 31 characters for a physical characteristic and a maximum of 255 characters for a logical characteristic. These characteristics are installation-specific.

Default: NOCHARACTERISTIC

NOTE

Specifies that a note is to be printed on the flag page of the report.

Limit: The note can be up to 255 characters long.

NOTIFY|NONOTIFY

NOTIFY indicates that a message is to be sent to the user's terminal when the report has been printed. NONOTIFY indicates that the user is not to be notified.

Default: NONOTIFY

OPERATOR|NOOPERATOR

OPERATOR specifies that a message is to be sent to the system operator. The report is not printed until the operator responds to the message. NOOPERATOR indicates that no message is to be sent to the system operator.

Limit: The message can be up to 255 characters long.

Default: NOOPERATOR

PRIORITY n

Specifies the job's scheduling priority.

Limit: 255

QUEUE queuename

Specifies the name of the queue in which the printed report is to be queued, and also determines the device where the report is to be printed.

Default: The default printer device is SYS$PRINT.

RESTART|NORESTART

RESTART indicates that the report is to be restarted after a crash or after the DCL command STOP/QUEUE/REQUEUE has been issued. NORESTART indicates that the report is not to be restarted.

Default: RESTART

TRAILER|NOTRAILER

TRAILER indicates that a trailer page is to be printed at the end of the report. NOTRAILER indicates that a trailer page is not to be printed at the end of the report.

Default: The system default is TRAILER.

Discussion

You can override the default options that are in effect for SHOW statements with the SET statement.

For example, a report without headings, page numbers, and other aspects of the default report format can be produced with the statement

```
> SET COMPRESSED
```

Examples

To choose a different dictionary without exiting from QSHOW, enter SET DICTIONARY, followed by the name of the file containing the dictionary to be reported, as in

```
> SET DICTIONARY NEWDICT
```

By default, QSHOW reports appear on the terminal. To send QSHOW reports to the printer, use the REPORT DEVICE option of the QSHOW statement as follows:

```
> SET REPORT DEVICE PRINTER
```

With this statement in effect, all reports from subsequent SHOW statements are sent to the printer. To change back to terminal output, enter

```
> SET REPORT DEVICE TERMINAL
```

SHOW

Reports dictionary definitions.

Syntax

SHOW option

Options

SHOW Options		
ALL	APPLICATION SECURITY	BASES
DATABASES	DESCRIPTION OF	ELEMENTS|FILES|USAGES
OPTIONS	PORTS	RECORDS
SECURITY	SUBFILE	SUBFILE RECORD
SYSTEMS	TRANSACTION	USAGES
USERS		

ALL [SUMMARY|DETAIL]

Reports the entire contents of the dictionary.

SUMMARY|DETAIL

Controls the level of detail that's provided in the report of the dictionary contents.

Default: DETAIL

[APPLICATION] SECURITY [SUMMARY|DETAIL] (UNIX, Windows)

For information about security on **MPE/iX**, see (p. 152), and on **OpenVMS**, see (p. 149).

Reports all application security classes and the UICs that belong to them.

Limit: The SET SECURITY statement must be in effect in order to report application security classes. The SET SECURITY and SHOW APPLICATION SECURITY statements are only available to dictionary owners. (The dictionary owner is the person who created the data dictionary.)

[APPLICATION|DICTIONARY] SECURITY [SUMMARY|DETAIL] (OpenVMS)

For information about security on **MPE/iX**, see (p. 152), and on **UNIX/Windows**, see (p. 152).

For PDC dictionaries, specifies and reports application security classes. For PHD dictionaries, specifies and reports application and dictionary security classes. If neither the APPLICATION nor the DICTIONARY keyword is specified, both are displayed.

Limit: The SET SECURITY option must be in effect for security to be reported. If it is not, QSHOW issues a warning. The APPLICATION|DICTIONARY SECURITY option can only be used by users with application or dictionary manager status, and by the dictionary creator.

BASES [DETAIL|SUMMARY] (MPE/iX)

Reports on IMAGE databases in QDD and QDDR dictionaries only. Only IMAGE databases are reported.

Default: SUMMARY

DATABASES [name-option] [DETAIL|SUMMARY]

Shows the database names (including IMAGE) from databases that are declared with either the FILE or DATABASE statements. It also shows PowerHouse file names included in the dictionary. Tables, views, and record-structures are only displayed if the DETAIL option is used.

name-option

If you don't specify a name-option, QSHOW reports the descriptions for all definitions of the specified entity.

For more information about name-options, see (p. 134).

DETAIL|SUMMARY

Controls the level of detail that QSHOW provides about the given entity. SHOW DATABASES DETAIL also includes the dictionary status of the [NO] CRITICAL ITEM UPDATE option on the FILE statement:

```
CRITICAL ITEM UPDATE : ON
```

QSHOW cannot tell you whether the database has critical item update enabled since the database administrator has the ability to enable or disable critical item update externally. The actual status is verified by QUICK or QTP when they access the database.

Default: DETAIL if you specify a name-option; otherwise SUMMARY.

DESCRIPTION OF ASC [name-option] (UNIX, Windows)
DESCRIPTION OF ELEMENTS [name-option]
DESCRIPTION OF FILES [name-option]
DESCRIPTION OF RECORDS [name-option]
DESCRIPTION OF USAGES [name-option]

Reports descriptions of specified entities.

ASCS (UNIX, Windows)

Reports application security classes.

ELEMENTS

Reports element descriptions.

FILES

Reports file descriptions. This option recognizes relational tables.

RECORDS

Reports record-structure descriptions.

USAGES

Reports usage descriptions.

name-option

Specifies that QSHOW is to report the descriptions for a particular subset of the specified entity.

If you don't specify a name-option, QSHOW reports the descriptions for all definitions of the specified entity.

For more information about name-options, see (p. 134).

ELEMENTS|FILES|USAGES [name-option] [SUMMARY|DETAIL]

Generates a report of entity definitions.

ELEMENTS

Reports element definitions.

FILES

Shows the names of record structures, tables, or views from files and databases that are declared with FILE or DATABASE statements in PDL.

USAGES

Reports usage definitions.

name-option

Specifies that QSHOW report a specific subset of the specified entity. If you don't specify a name-option, QSHOW reports all definitions for the specified entity.

For more information on name-options, see (p. 134).

SUMMARY|DETAIL

Controls the level of detail that QSHOW provides about the given entity.

Default: DETAIL if you specify a name option; otherwise, SUMMARY.

OPTIONS (MPE/iX, UNIX, Windows)

Reports the system-wide options as specified in the dictionary.

PORTS (MPE/iX, UNIX)

Reports the terminal types assigned to ports.

RECORDS [name-option] [IN file|database] [SUMMARY|DETAIL]

Reports record-structure definitions from a data dictionary or reports relational tables or views as PowerHouse record-structures. For INDEXED files, the report shows the segments of a multiple-segment index. For relational databases, record detail reports include the names of columns and indexes that each relation contains. If you are running QSHOW using the default program parameters, you must use the IN database qualifier when you name particular relations. Normally, only user-defined relations are reported. Relational system tables are not reported.

The output of the SHOW RECORDS statement displays the letter "r" to indicate that column values are required. The following table shows how null value specifications affect the QSHOW display.

Database column specification	PDL FILE or DATABASE option	QSHOW SHOW RECORD display
nulls allowed	NULL VALUES NOT ALLOWED	"r"
nulls not allowed	NULL VALUES NOT ALLOWED	"r"
nulls allowed	NULL VALUES ALLOWED	
nulls not allowed	NULL VALUES ALLOWED	"r"

name-option

Specifies that QSHOW is to report a specific subset of the record-structure. If you don’t specify a name-option, QSHOW reports all definitions for the record-structure.

For more information on name-options, see (p. 134).

file

A collection of data records. A file can contain more than one record-structure.

database

The name of a relational database declared in the data dictionary.

SUMMARY|DETAIL

Controls the level of detail that QSHOW provides about the record-structure.

SECURITY [SUMMARY|DETAIL] (MPE/iX)

For security on UNIX/Windows, see (p. 149). For security on OpenVMS, see (p. 149).

Reports all the application security classes and the logonids that belong to them.

For this report to be displayed, you must enter the SET SECURITY statement.

SET SECURITY has no effect unless you are the creator of the dictionary or have System Manager or Account Manager capability.

SUMMARY|DETAIL

Controls the level of detail that QSHOW provides about the given entity.

SUBFILE filespec [ALL|ELEMENTS] [SUMMARY|DETAIL]
SUBFILE filespec RECORD

Reports on a subfile from its minidictionary. For QSHOW to create reports on a subfile, the subfile must already exist. The subfile must be a permanent file, or it must have already been created by PowerHouse in your current session.

Default: ALL

filespec

Names the subfile to be reported.

ALL

Reports the record-structure of the subfile, as well as each element the subfile contains. SUMMARY and DETAIL options control the amount of reported information.

Default: SUMMARY

ELEMENTS

Reports each element in the subfile. The SUMMARY and DETAIL options control the amount of detail in the report.

Default: SUMMARY

SUMMARY|DETAIL

Controls the level of detail that QSHOW provides about the given entity.

RECORD

Reports subfile record-structures.

SYSTEM

Reports the system-wide options as specified in the dictionary.

TRANSACTION [name-option] [SUMMARY|DETAIL]

Reports transaction definitions.

USAGES [name-option] [SUMMARY|DETAIL]

Reports usage definitions.

USERS

Reports all the application security classes and the user ID codes that belong to them.

For the user ID to be displayed, you must enter the SET SECURITY statement. SET SECURITY has no effect unless you are the creator of the dictionary (**MPE/iX, UNIX, Windows**).

Limit: SHOW USERS can only be used by users with application manager, dictionary manager, or creator status (**OpenVMS**).

Name-Options

Name-options restrict reporting to a single name or a set of names. If you don't specify a name-option, QSHOW reports all definitions of the specified entity.

The name-options are namelist, PATTERN, and SOUNDEX.

namelist

Retrieves the names identified by a combination of single names, lists of names, and ranges of names.

The general form is:

name [TO name] [,name [TO name]]...

Multiple name-sets must be separated by commas.

Limit: 10 names per set.

PATTERN string

Retrieves names that match the string of characters specified in the pattern string. For more information, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

SOUNDEX(string[,n])

For more information, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Discussion

The SHOW statement generates reports on dictionary definitions. Particular entities or groups of entities can be included in a report. The following can be specified for elements and files:

- individual names
- lists and ranges of names
- pattern matching
- SOUNDEX matches

With the SHOW statement you can report the entire contents of the dictionary or particular entities as indicated in the following list:

To report	Enter
entire dictionary	SHOW ALL
IMAGE databases in QDD and QDDR (**MPE/iX**)	SHOW BASES
databases/files	SHOW DATABASES, SHOW FILES
tables and views	SHOW DATABASES
tables, views, and record-structures	SHOW FILES
application security	SHOW SECURITY
dictionary security (**OpenVMS**)	SHOW SECURITY
records and indexes	SHOW RECORDS
elements	SHOW ELEMENTS
descriptions	SHOW DESCRIPTION

To report	Enter
system-wide standards	SHOW OPTIONS (**MPE/iX, UNIX, Windows**) SHOW SYSTEM
subfiles	SHOW SUBFILES
element usages	SHOW USAGES
user ID codes	SHOW USERS

Controlling Reported Information

The SUMMARY and DETAIL options control the amount of reported information. When you don't specify SUMMARY or DETAIL as part of a SHOW statement, QSHOW assumes the following:

Statement	Default
SHOW ALL	DETAIL
SHOW entity	SUMMARY
SHOW entity namelist	DETAIL
SHOW entity PATTERN string	SUMMARY
SHOW entity SOUNDEX (string)	SUMMARY

These defaults simplify the most common reporting requirements.

Examples

To report all files, enter

```
> SHOW FILES
```

Reporting Entities by Namelist

Any combination of names, lists of names, and ranges of names is referred to in QSHOW as a namelist. Namelists work exactly the same way for elements and files. To report specific files, list them by name, as in

```
> SHOW FILES CUSTOMER, PROJECTS, EMPLOYEES
```

To report a range of names with a statement, enter

```
> SHOW FILES C TO F, PROJECTS
```

This statement reports files with names that begin with C through to those that begin with F, as well as the file PROJECTS. The same techniques can be used to report element, record, and usage names.

Reporting Entities by Pattern Matches

As an alternative to searching by namelist, you can make more sophisticated searches for names using patterns and SOUNDEX matches.

Searching for names by pattern can be accomplished by using the PATTERN option and a pattern string. For example, the statement,

```
> SHOW ELEMENTS PATTERN "@POS@"
```

reports all elements containing POS in their names. Pattern matching in QSHOW follows the same rules as in other PowerHouse components. The only exception is that patterns used in QSHOW are automatically upshifted. For example, you can enter "@pos@" or "@POS@" and get the same result. For more information, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Reporting Names Using SOUNDEX Codes

SOUNDEX matches words with similar phonetics. For example, the statement

```
> SHOW ELEMENTS SOUNDEX("LASTNAME")
```

retrieves names such as LAST_NAME, LST_NAME, LST_NM, and LSTNM.

For more information, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Using SHOW DESCRIPTION

The SHOW DESCRIPTION statement allows you to report any file or element descriptions in the dictionary. To report a selection of element descriptions, use a namelist, as in

```
> SHOW DESCRIPTION OF ELEMENTS E TO H, CITY
```

You can follow the keywords, ASCS, ELEMENTS, FILES, RECORDS, or USAGES, with namelists, patterns, and SOUNDEX matches.

You can control the amount of detail reported with the keywords, SUMMARY and DETAIL. A SUMMARY report is a one-line-per entity summary of important information. A DETAIL report is a full feature report that summarizes the entity in several lines and includes cross-references.

USE

Processes QSHOW source statements contained in a file.

Syntax

USE filespec [option]

filespec

Names the file that contains the QSHOW source statements you want to use.

If the file doesn't exist, QSHOW issues an error message.

Options

The options are DETAIL, NODETAIL, LIST and NOLIST.

DETAIL|NODETAIL

DETAIL writes the contents of the file to QSHOW's source statement save file, qshosave, rather than just the USE statement itself. NODETAIL writes just the USE statement to the temporary save file rather than the contents of the file being used.

Default: DETAIL

LIST|NOLIST

LIST displays the statements as they are processed; NOLIST doesn't.

Default: LIST

Discussion

The USE statement instructs QSHOW to process the named file for statement input.

QSHOW reads and interprets each statement as if it had been entered from the terminal. By entering USE statements, you can run standard reports that you've prepared in advance.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the USE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Nesting USE Statements

A file referenced in a USE statement can itself contain other USE statements. USE files can be nested to a maximum of 20 levels. Permanent files and record-structures containing valid source code can be included at any time provided they are consistent with QSHOW syntax and structure.

Chapter 5: QUTIL Statements

Overview

QUTIL is a utility that creates and deletes non-relational files and databases.

QUTIL uses the syntax and language rules that are discussed in Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

For more about running QUTIL, see Chapter 1, "*Running PowerHouse*", in the *PowerHouse Rules* book.

Summary of QUTIL Statements

The following table summarizes the purpose of each QUTIL statement:

Statement	Purpose
CREATE	Creates files.
DELETE	Deletes files.
EXIT	Terminates QUTIL.
QSHOW	Runs QSHOW from QUTIL.
QUIT	Terminates QUTIL.
REVISE	Edits files from within QUTIL.
SAVE	Saves QUTIL source statements in a file.
SET	Changes the settings for a QUTIL session.
USE	Processes QUTIL statements contained in a file.

CREATE (MPE/iX, UNIX, Windows)

For CREATE (**OpenVMS**), see (p. 162).

Creates files, IMAGE databases, and Eloquence.

Syntax

CREATE ALL

CREATE BASE name [base-option]...[[,] name[base-option]...]...

CREATE FILE name [file-option]...[[,] name[file-option]...]...

Options

ALL

Creates all files, IMAGE databases, and Eloquence databases declared in the dictionary.

Limit: Files declared with the NOCREATE attribute, and IMAGE or Eloquence databases whose datasets are declared with the NOCREATE attribute are not created.

BASE

Creates the named IMAGE or Eloquence databases. For IMAGE, use the actual database name. For Eloquence, use the logical name declared in the PDL FILE OPEN clause. For example, if the PDL code for an IMAGE database is

```
> FILE mydb ORGANIZATION DATABASE OPEN testdb.mygroup.myaccount
```

then the QUTIL code to create the database would be

```
> CREATE BASE testdb
```

However, if the PDL code for an Eloquence database is

```
> FILE mydb ORGANIZATION DATABASE OPEN server001:eloqdb/testdb
```

then the QUTIL code to create the database would be

```
> CREATE BASE mydb
```

FILE

Creates all the named files defined in the dictionary, even if some of those files are specified with the NOCREATE attribute in the dictionary.

Limit: Not valid for relational databases.

base-option (MPE/iX only)

Allows you to override certain attributes of datasets that were specified in the dictionary. The base options are BLOCKMAX and CAPACITY.

BLOCKMAX n

Specifies the maximum block length (in words) of any dataset in the database.

Limit: This option must be used when the record length of an IMAGE dataset is larger than 512 words. The maximum block length can be from 128 to 2048 words.

CAPACITY n

Specifies the maximum number of records (n) in each dataset in the database. The capacity can be any number within the limits set by IMAGE. This option overrides the individual capacities specified for each dataset in the dictionary. Since capacity can be specified in QUTIL, and QSHOW reflects the capacity specified in the dictionary, the values shown by QSHOW may not be the actual capacity of the dataset.

file-option (MPE/iX only)

Allows you to override certain attributes of files that were specified in the dictionary. The file options are BLOCKING FACTOR, CAPACITY, and TEMPORARY.

BLOCKING [FACTOR] n

Specifies the number of records (n) per block. This option overrides any blocking factor specified in the dictionary.

CAPACITY n

Specifies the maximum number of records (n) in the file. The capacity can be any number within the limits set by the operating system. This option overrides the capacity specified in the dictionary. Since capacity can be specified in QUTIL, and QSHOW shows the capacity specified in the dictionary, the value shown in QSHOW may not be the actual capacity of the file.

TEMPORARY

Creates a temporary file rather than a permanent file.

Discussion

The CREATE statement builds files, IMAGE databases, and Eloquence databases declared in the dictionary. If the file already exists, you are prompted for permission to create a new file. QUTIL doesn't attempt to create files to which you don't have write access (as declared in the dictionary).

If more than one FILE statement specifies the same open name for files of the same type, QUTIL applies the following rules:

1. If one of the files has the CREATE option specified, that file is created.
2. If one of the files has the NOCREATE option specified, that file is not created.
3. The first file in alphabetical order is created.

To ensure that QUTIL creates the right file, use either the CREATE option on one FILE statement only, or the NOCREATE option on all but the one file you want to create.

To specify more than one file or database in a single CREATE statement, separate each file name or database name with a space or a comma.

If only some of the datasets in a database are declared in the dictionary or some datasets are declared with the NOCREATE option, don’t use QUTIL’s CREATE statement to build the database. QUTIL attempts to create a database from the definitions in the dictionary, even if datasets are missing. To prevent QUTIL from trying to build a partial database, specify the NOCREATE option on each RECORD statement for the database.

For more information on creating files and databases, see the discussion of the CREATE|NOCREATE options of the FILE and RECORD statements on (p. 62) and (p. 62), respectively.

Each name that follows the FILE keyword is interpreted as a file name. Similarly, each name that follows the BASE keyword is then interpreted as a database name.

To combine file and database names in the same statement, follow the FILE keyword with the file names, and the BASE keyword with the database names, as in

```
> CREATE BASE INVEN FILE INVEN-AUDIT INEVEN-TXN
```

This statement creates a database named INVEN, and two files named INVEN-AUDIT and INVEN-TXN.

Creating an IMAGE database

To create an IMAGE database, QUTIL generates the necessary schema and invokes the HP programs, DBSCHEMA and DBUTIL. As the IMAGE schema is processed by DBSCHEMA, the DBSCHEMA output is displayed on your terminal. When DBSCHEMA is finished, QUTIL invokes the DBUTIL program to create the database. If more than one database is being created, DBSCHEMA runs separately for each database. DBUTIL runs once for all the IMAGE databases.

The record layout for an IMAGE dataset must be defined to have an even storage length, since IMAGE does not support odd length records. Although QUTIL always creates datasets with even-length records by adding an extra byte to an odd-length record, the resulting IMAGE schema cannot be used by PowerHouse because the actual record length doesn't match the length recorded in the dictionary. QUTIL issues an error message to indicate this problem.

[NO] CRITICAL ITEM UPDATE option on the FILE statement (MPE/iX)

QUTIL generates the following DBUTIL syntax:

```
SET BASENAME CIUPDATE=ALLOWED
```

The ALLOWED option indicates that the [NO] CRITICAL ITEM UPDATE option, specified on the FILE statement in PDL, is allowed if it is enabled within the database.

The other options of the DBUTIL SET CIUPDATE statement are DISALLOWED and ON.

Limit: If you are creating a database with QUTIL, PowerHouse will not support the ON option of the DBUTIL SET CIUPDATE statement. This is to avoid data integrity problems that could arise as a result of using RUN commands or DO EXTERNAL commands to access programs that do not support critical item update.

Creating a KSAM file

To create a KSAM file, QUTIL generates KSAMUTIL commands for the file creation, and then it invokes the HP program, KSAMUTIL. If more than one KSAM file is being created, KSAMUTIL runs once for all the KSAM files.

Creating MPE and KSAMXL files

To create MPE or KSAMXL files, QUTIL uses the MPE BUILD command.

Datatype Mapping Tables

PowerHouse makes few restrictions on datatype usage for keys and indexes. File systems, on the other hand, are much more restrictive, so PowerHouse maps its datatypes to the best match available when the file is generated using QUTIL. Also see "DISAM Data Storage" in *PowerHouse Rules*.

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

PowerHouse Datatype	IMAGE Datatype	KSAM Datatype	C-ISAM/DISAM Datatype
PHDATE	K	BYTE	CHARTYPE
VARCHAR[4]	--	BYTE	CHARTYPE
ZDATE	X6	BYTE	CHARTYPE
ZONED SIGNED[3]	Z	BYTE	CHARTYPE
ZONED UNSIGNED[3]	Z	NUMERIC	CHARTYPE

[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.

CREATE (OpenVMS)

For CREATE (**MPE/iX, UNIX, Windows**), see Creates files (p. 158).

Syntax

CREATE ALL

CREATE FILE filename[[,]filename]...

Options

ALL

Creates all files declared in the dictionary.

Limit: Files declared with the NOCREATE attribute are not created.

FILE

Creates all the named files defined in the dictionary, even if some of those files are specified with the NOCREATE attribute in the dictionary.

Limit: Not valid for relational databases.

Discussion

The CREATE statement builds files declared in the dictionary. If the file already exists, you are prompted for permission to create a new file. QUTIL doesn't attempt to create files to which you don't have write access (as declared in the dictionary).

If more than one FILE statement specifies the same open name for files of the same type, QUTIL applies the following rules:

1. If one of the files has the CREATE option specified, that file is created.
2. If one of the files has the NOCREATE option specified, that file is not created.
3. The first file in alphabetical order is created.

To ensure that QUTIL creates the right file, use either the CREATE option on one FILE statement only, or the NOCREATE option on all but the one file you want to create.

To specify more than one file in a single CREATE statement, separate each file name with a space or a comma.

If FDL is set, then an FDL file is created along with the data file with the name <*filename*>.fdl.

Datatype Mapping Tables

PowerHouse makes few restrictions on datatype usage for keys and indexes. File systems, on the other hand, are much more restrictive, so PowerHouse maps its datatypes to the best match available when the file is generated using QUTIL.

PowerHouse	RMS ISAM
CHARACTER	character string
DATE	2-byte unsigned integer
DATETIME	8-byte unsigned integer
FLOAT[5]	not permitted
FREEFORM[1]	character string

PowerHouse	RMS ISAM
G-FLOAT[5]	not permitted
INTEGER SIGNED[2]	2-byte signed integer 4-byte signed integer 8-byte signed integer
INTEGER UNSIGNED	2, 4, 8-byte unsigned integer
INTERVAL[5]	not permitted
JDATE	2-byte unsigned integer
PACKED SIGNED [3]	packed
PACKED UNSIGNED [3]	packed
PHDATE	2-byte unsigned integer
VARCHAR[4]	character string
VMSDATE	8-byte unsigned integer
ZDATE	character string
ZONED SIGNED[3]	character string
ZONED UNSIGNED[3]	character string
ZONED NUMERIC	character string

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

[2] CHARTYPE does not process 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.

[5] RMS ISAM allows only certain datatypes as index segments.

DELETE (MPE/iX)

For DELETE (**OpenVMS, UNIX, Windows**), see (p. 165).

Deletes files and IMAGE databases.

Syntax

DELETE ALL

DELETE BASE name [[,]name]...

DELETE FILE filespec [[,]filespec]...

Options

ALL

Deletes all files and databases declared in the dictionary.

Limit: Files declared with the NOCREATE attribute, and the databases whose datasets are all declared with the NOCREATE attribute, are not deleted.

BASE

Deletes the named databases.

FILE

Deletes all named files specified in the dictionary, even if some of the files have the NOCREATE attribute.

Limit: Not valid for relational databases.

Discussion

The DELETE statement deletes files and databases declared in the dictionary. QUTIL does not delete files and databases to which you don't have write access. Similarly, if you enter DELETE ALL, QUTIL won't delete files specified as NOCREATE in the dictionary, or databases whose datasets are declared in the dictionary as NOCREATE. When indexed files are deleted, key files are deleted along with the data files.

When a database is deleted, all its datasets are deleted, even those not declared in the dictionary. When KSAM files are deleted, their key files are deleted along with the data files.

To specify more than one file in a single DELETE statement, separate each filename with a space or comma. DELETE verifies that the file being deleted matches the dictionary definitions. Use DELETE before changing the dictionary definitions.

Each name that follows the FILE keyword is interpreted as a file name. Similarly, each name that follows the BASE keyword is interpreted as a database name.

To combine file and database names in the same statement, follow the FILE keyword with the file names, and the BASE keyword with the database names, as in

```
> DELETE BASE INVEN FILE INVEN-AUDIT INVEN-TXN
```

This statement deletes a database named INVEN, and two files named INVEN-AUDIT and INVEN-TXN.

DELETE verifies that the file being deleted matches the dictionary definitions. Use DELETE before changing the dictionary definitions.

If you want to change the dictionary definition of a file but keep the data, save the data in a subfile before making any changes. Once you have recreated the file with the new dictionary definitions, you can restore the data to the new file.

DELETE (OpenVMS, UNIX, Windows)

For DELETE (**MPE/iX**), see (p. 164).

Deletes files.

Syntax

DELETE ALL

DELETE FILE filespec [[,]filespec]...

Options

ALL

Deletes all files declared in the dictionary.

Limit: Files declared with the NOCREATE attribute are not deleted.

FILE

Deletes all named files specified in the dictionary, even if some of the files have the NOCREATE attribute.

Limit: Not valid for relational databases.

Discussion

The DELETE statement deletes files declared in the dictionary. QUTIL does not delete files to which you don't have write access. Similarly, if you enter DELETE ALL, QUTIL won't delete files specified as NOCREATE in the dictionary. When indexed files are deleted, key files are deleted along with the data files.

To specify more than one file in a single DELETE statement, separate each filename with a space or comma. DELETE verifies that the file being deleted matches the dictionary definitions. Use DELETE before changing the dictionary definitions.

If you want to change the dictionary definition of a file but keep the data, save the data in a subfile before making any changes. Once you have recreated the file with the new dictionary definitions, you can restore the data to the new file.

OpenVMS: If FDL (File Definition Language) is set and an associated FDL file is found, it is deleted along with the data file.

EXIT

Terminates QUTIL.

Syntax

EXIT

Discussion

The EXIT statement terminates QUTIL and returns control to the invoking program.

EXIT and QUIT are interchangeable.

QSHOW

Runs QSHOW from QUTIL.

Syntax

QSHOW

Discussion

The QSHOW statement initiates a QSHOW session from within QUTIL. For more information about QSHOW, see Chapter 4, "QSHOW Statements". When you EXIT from QSHOW, you are returned to your QUTIL session.

QUIT

Terminates QUTIL.

Syntax

QUIT

Discussion

The QUIT statement terminates QUTIL and returns control to the invoking program.

QUIT and EXIT are interchangeable.

REVISE

Edits files from within QUTIL.

Syntax

**REVISE [*|filespec [DETAIL|NODETAIL] [LIST|NOLIST]
[USE|NOUSE]]**

*

Signifies that the current save file is to be edited. The save file is qutlsave (**MPE/iX**) and qutlsave.qus (**OpenVMS, UNIX, Windows**).

This source statement save file is a temporary file that the PowerHouse component opens at the beginning of a session. All statements except the SAVE statement are recorded in this file as you enter them. The asterisk is required only if you change the default options while editing the source statement file.

filespec

Names an existing permanent file. If this file does not contain component statements, use the NOUSE option. This ensures that the component will not try to execute the file when you exit from the system editor.

DETAIL|NODETAIL

DETAIL copies the contents of the revised file into the QUTIL save file when you exit from the system editor. NODETAIL puts a USE statement with the filespec of the revised file into the QUTIL save file. If you are revising a permanent file with USE and NODETAIL in effect, then QUTIL adds the following statement to the save file:

```
> USE ORDERS NODETAIL
```

where ORDERS is the name of the revised file.

Limit: The NODETAIL option is not valid with the current source statement save file.

Default: DETAIL

LIST|NOLIST

LIST displays each statement from the revised file as they are executed. NOLIST processes the statements without displaying them.

Default: LIST

USE|NOUSE

USE executes the revised statements when you exit from the system editor. NOUSE does not. NOUSE returns you to the PowerHouse component at the point from which you left it.

Default: USE

Discussion

The REVISE statement indicates which file is to be edited and, optionally, how the revised file is to act upon reentering QUTIL.

Defaults:

MPE/iX:	By default, the REVISE statement chooses EDITOR/3000 as the system editor. You can change this by using a file equation for COGEDITR.
OpenVMS:	REVISE uses the editor defined in PHEDIT. You can change the default editor by changing the setting of the PHEDIT symbol. If PHEDIT is not defined, REVISE uses the system editor, EDT.

UNIX, Windows:	REVISE uses the editor defined in PHEDIT. If PHEDIT is not defined, the system checks the environment variable, EDITOR. If you have not defined either of these variables, the REVISE statement fails.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the REVISE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

SAVE

Saves QUTIL source statements in a file.

Syntax

SAVE filespec[CLEAR]

Options

filespec

Names a permanent file in which to save the QUTIL statements.

CLEAR

Deletes the contents of QUTIL's temporary source statement save file after the contents are copied to the permanent file named by filespec. The save file is qutlsave (**MPE/iX**) and qutlsave.qus (**OpenVMS, UNIX, Windows**).

Discussion

The SAVE statement copies the contents of the QUTIL save file to a permanent file. The save file is a temporary file that QUTIL opens at the beginning of a session. All QUTIL statements (except for the SAVE statement) are recorded into the save file as they are entered. The saved contents can be used as a source file for documentation and future changes, or as a working file to be modified using a text editor. The saved statements can also be processed by QUTIL with the USE statement.

The CLEAR option deletes the contents of QUTIL save file after copying the contents to a permanent file. This allows you to enter and then save a new set of QUTIL statements in the same session. (To clear the save file without saving its contents, use the SET SAVE CLEAR statement.)

SET

Changes the settings for a QUTIL session.

Syntax

SET option [option]...

Options

DICTIONARY filespec [TYPE PHD|PDC]

Names the dictionary to be used for the current QUTIL session. SET DICTIONARY can be used any number of times in a single session, and is helpful when more than one dictionary is being referenced.

MPE/iX: SET DICTIONARY does not affect file equations directing PowerHouse to a certain dictionary. However, SET DICTIONARY does override the file equations for the session.

Default: QUTIL references the dictionary you specified before entering PowerHouse.

[TYPE PHD|PDC] (OpenVMS)

Specifies the default dictionary type. If the TYPE option is specified in a PowerHouse component, it applies to subsequent SET DICTIONARY statements in the component.

When searching for a dictionary, PowerHouse limits searches to the dictionary type specified by the TYPE option. If the TYPE option is not specified, PowerHouse searches first for a PHD dictionary, then a PDC dictionary.

Default: PHD

FDL|NOFDL (OpenVMS)

Specifies whether to create or delete FDL (File Definition Language) files. FDL files have the form filename.fdl. If FDL is specified

- the CREATE statement creates the FDL file
- the DELETE statement deletes the FDL file

Default: NOFDL

GO|NOGO (MPE/iX)

When GO is in effect, QUTIL creates and deletes files and databases as each CREATE and DELETE statement is entered. When NOGO is in effect, the create and delete actions are deferred. When used in conjunction with SOURCE, NOGO gives you the opportunity to edit the source file created by SOURCE and to run the creation or deletion as a batch job.

Limit: NOGO can only be used when SOURCE is in effect.

Default: GO

STATISTICS|NOSTATISTICS

STATISTICS reports details about the file being created, such as, record size, indexes, and physical location. NOSTATISTICS does not.

Default: NOSTATISTICS

SOURCE [filespec]|NOSOURCE (MPE/iX)

SOURCE creates a file containing the statements required to perform the QUTIL functions you subsequently enter. NOSOURCE resets the SOURCE option, closing any currently open source file. The job that is created with SOURCE contains a dummy job control statement that must be modified before you can stream the file.

With SET SOURCE in effect, the contents of the source file provide documentation of your activity.

With SET NOGO in effect, the source file can be edited before being run, then streamed as a job.

IF SET SOURCE is specified without a filespec, a file named QUTLSRC is created.

Default: NOSOURCE when SET GO is in effect; SOURCE when SET NOGO is in effect.

VERIFY[ALL|DELETE]|NOVERIFY

VERIFY requests authorization to proceed with processing. NOVERIFY instructs QUTIL not to prompt for permission to proceed during processing. When VERIFY ALL is specified, QUTIL prompts you for permission to proceed before creating or deleting a file. If VERIFY DELETE is specified, QUTIL prompts you for permission to proceed before deleting a file or database (**MPE/iX**).

If SET NOGO and SET SOURCE are in effect, the VERIFY prompts will govern which statements are put into the source file for later execution. For example, if VERIFY ALL is in effect, QUTIL displays prompts like

```
DATA BASE INVEN Already exists.
   O.K. to generate delete and create statements?
```

If you respond with a Yes, the commands to delete and then create the file or database (**MPE/iX**) are put into the source file.

Default: VERIFY DELETE

Discussion

SET statements override default options normally in effect during a QUTIL session.

USE

Processes QUTIL statements contained in a file.

Syntax

USE filespec[DETAIL|NODETAIL] [LIST|NOLIST]

Options

filespec

Names a file that contains QUTIL source statements.

DETAIL|NODETAIL

DETAIL copies the contents of the source file to QUTIL's temporary save file rather than just the USE statement itself. NODETAIL copies just the USE statement. For more information about QUTIL's temporary save file, qutlsave (**MPE/iX**), or qutlsave.qus (**OpenVMS, UNIX, Windows**), see (p. 171).

Default: NODETAIL

LIST|NOLIST

LIST displays each QUTIL statement contained in the file as they are executed. NOLIST processes the statements without displaying them.

Default: LIST

Discussion

The USE statement instructs QUTIL to read the named file for statement input. QUTIL reads and interprets each statement as if it had been entered from the terminal. The file can contain other USE statements.

Limit: USE statements can be nested to a maximum of 20 levels.

The **procloc** parameter affects how PowerHouse uses unqualified file names that are specified in the USE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Chapter 6: ITOP Utility

Overview

In this chapter, you will learn how to:

- use the IMAGE to PDL conversion utility (ITOP)
- handle conversion issues

Converting to PDL

The ITOP utility is used to generate PDL statements directly from an existing IMAGE database. You can create a basic dictionary from the IMAGE database, shortening the time it takes to design a full dictionary.

The ITOP utility prompts for

- a dictionary title
- application security class names and associated logon IDs
- database information

In response to this information, ITOP

- reads the IMAGE root file for each database
- generates PDL dictionary design statements for a single dictionary
- places the statements in an automatically created MPE file named PDLSRC

To start an ITOP session

1. Enter

   ```
   :ITOP
   ```

   Note: If the dictionary design is likely to contain more than 5000 lines, use the **parm** program parameter to specify the maximum number of lines. For example:

   ```
   :ITOP PARM=8000
   ```
2. Enter a dictionary title.
3. Enter an application security class name.
4. Enter the logon IDs of the users belonging to the application security class.

   Note: You can enter the same logon ID in more than one application security class.
5. Repeat steps 3 and 4 for each application security class.
6. Enter the name of the IMAGE database.

   Include the database group and account if they are different from the current group and account.
7. Enter the database password.

   Enter either Y (Yes) or N (No) depending on whether you want ITOP to duplicate the access granted by the IMAGE passwords in the dictionary design.

If you select...	Then...
N (No)	Skip to step 8.

If you select...	Then...
Y (Yes)	ITOP prompts for the IMAGE password corresponding to each application security class. For example MANAGER : SESAME CLERK : LOOK In this example, all users belonging to the application security class MANAGER will have the same access that IMAGE grants users with the password SESAME.

8. Repeat steps 6 through 7 for each database.

Issues for Consideration

When using ITOP to generate PDL from an IMAGE database, you need to consider the following:

- IMAGE security
- duplicate names
- date elements

IMAGE Security

ITOP attempts to duplicate existing IMAGE security in the PDL statements that it creates. ITOP relies on a one-to-one correspondence between PDL application security class names and IMAGE passwords. ITOP generates the appropriate PERMIT statement to reflect the access specified in the IMAGE root file.

ITOP prompts for the information that is necessary for a dictionary design that can't be found in an IMAGE root file. To duplicate the existing IMAGE security for any of the databases, you must enter information at the application security class and password prompts. If you do not want to duplicate existing IMAGE security, you can ignore those prompts.

Duplicate Names

While creating PDL statements, ITOP indicates design errors. The most common errors are duplicate names, which can result when more than one database is used to create a single dictionary design. At the end of the ITOP session, you will receive a message indicating duplicate file names.

Although ITOP can only flag duplicate file names, it does attempt to resolve duplicate element names. If two elements have the same name and identical attributes, including security, only one is included in the dictionary design. Because any element can be found in any number of files, this built-in mechanism for eliminating duplicate element definitions doesn't make any difference to the resulting dictionary design. If, however, two elements have the same name but their attributes aren't identical, ITOP adds both elements to the dictionary design and flags the second occurrence.

All errors must be corrected before PDL can create the dictionary. The PDLSRC file can be used as input to the PDL program.

Date Elements

ITOP does not use PowerHouse DATE datatypes in its generated PDL. To use DATE datatypes, you will have to modify the definition for IMAGE datatypes used as dates.

Refine Your Dictionary Design

You created the basic dictionary design with ITOP but you can enhance this skeletal design. The following list contains a sample of the types of changes you might want to make:

- add KSAM and MPE files
- add item redefinitions
- add or change implied decimal positions in element types
- add element descriptions, headings, and labels
- add new logon IDs
- restrict access of application security class to files and elements

Chapter 7: ETOP Utility

Overview

In this chapter, you will learn how to:

- use the Eloquence to PDL conversion utility (ETOP)
- handle conversion issues

Converting to PDL

The ETOP utility is used to generate PDL statements directly from an existing Eloquence database. You can create a basic dictionary from the Eloquence database, shortening the time it takes to design a full dictionary.

PowerHouse supports Eloquence on HP-UX and Windows through its "TurboIMAGE compatibiliby" interface, which makes it look (for the most part) like IMAGE. Thus ETOP is a variation on the ITOP utility (p. 175).

The ETOP utility prompts for

- a dictionary title
- application security class names and associated logon IDs (**Windows**, **UNIX**) or UICs (**UNIX**)
- database information

In response to this information, ETOP

- reads the Eloquence metadata for each database
- generates PDL dictionary design statements for a single dictionary
- places the statements in an automatically created UNIX or Windows file named pdlsrc.pdl.

To start an ETOP session

1. Enter
   ```
   $etop
   ```
   On Windows, ETOP can also be run from the Start menu.
2. You are prompted for a dictionary title. Enter one.
3. You are prompted with "Do you want to include Application Security?".

   If you enter N, you will skip to step 7. This is equivalent to entering <carriage return> in response to the first "USER NAME" prompt in ITOP.

   Windows: Entering Y results in a prompt to enter an application security class name.

   UNIX: Entering Y results in the prompt, "ASC ID METHOD? L=LOGONID, U=UIC". Enter L or U, resulting in a prompt to enter an application security class name.
4. Enter the first application security class name.
5. Enter one or more logon IDs (**Windows**, **UNIX**) or UICs (**UNIX**) of the users belonging to the application security class, one per line. Enter a blank line to end the list. UICs are entered with no surrounding square brackets, as two integers of 5 or fewer digits, separated by a comma and with no embedded blanks.

   Note: You can enter the same logon ID or UIC in more than one application security class.

   Note: While PDL accepts WEBLOGONID as a valid ASC ID METHOD, this is not supported by ETOP.
6. Repeat steps 4 and 5 for each application security class. Enter a blank line to end the list.
7. You are prompted for the open name of the first Eloquence database. Enter this, using the format `[[server][:service]/][Database]`.

8. You are prompted for the database password. This password must grant read/write access to all data sets in the database. The format of your response depends on the IMAGE emulator being used. For Eloquence, this must be a valid Eloquence [password]/user combination. Enter this.
9. If the database open name in step 7 is not a valid PowerHouse name (for example, "remote/REMOTEDB"), you are prompted for a PowerHouse name to use. Enter one.

   If you responded with N in step 3 to "Do you want to include Application Security?", skip to step 14. Otherwise, go to step 10.
10. You are prompted with "Do you want application security for this data base?".

    If you enter N, you skip to step 14. Otherwise, you go to step 11.
11. You are prompted with "Include element level security?". Enter Y or N as desired.

    Eloquence does not support element level security. A negative response will prevent the generation of PERMIT statements that will be ignored. The prompt is made in case another IMAGE emulator does support element level security.
12. You are prompted next with "Include security for the UNKNOWN ASC?". Enter Y or N as desired.

    This prompt works around a difference between IMAGE and Eloquence. For IMAGE, ITOP opens the database as "user class 0" by passing zero in the password parameter to DBOPEN and generating PERMIT statements granting our UNKNOWN ASC any access that is available to user class 0. Eloquence does not have anything that directly corresponds to IMAGE's user class 0. The closest it comes is a predefined Eloquence user called "public", which may or may not be password protected. If this public user has no access to anything in the database, an attempt to open the database as the public user fails. If this is the case, attempts to generate security related to our UNKNOWN class fails, so the user will want to respond negatively to this prompt.
13. You are prompted next for the database password corresponding to each ASC (including UNKNOWN, if you responded with Y in step 12). Enter these.

    These are passed to DBOPEN to open the database; the access granted to the associated ASC in the PDL source is that which is granted through this open of the database.

    Note: If DBSCHEMA source was used to create the Eloquence database, there may not be a one-to-one correlation between the PASSWORDs in the DBSCHEMA source and the Eloquence users. Each entry in the DBSCHEMA PASSWORDS section results in the creation of an Eloquence "group", and the "group" capabilities can then be granted to one or more Eloquence users. If more than one "group" is granted to a particular Eloquence user, that user will have access to everything that was granted though each of these groups.
14. Repeat steps 7 through 13 for each database.

Issues for Consideration

When using ETOP to generate PDL from an Eloquence database, you need to consider the following:

- Eloquence security
- duplicate names
- date elements

Eloquence Security

ETOP attempts to duplicate existing Eloquence security in the PDL statements that it creates. ETOP relies on a one-to-one correspondence between PDL application security class names and Eloquence passwords. ETOP generates the appropriate PERMIT statement to reflect the access specified in the Eloquence metadata.

ETOP prompts for the information that is necessary for a dictionary design that can’t be found in the Eloquence metadata. To duplicate the existing Eloquence security for any of the databases, you must enter information at the application security class and password prompts. If you do not want to duplicate existing Eloquence security, you can ignore those prompts.

Duplicate Names

While creating PDL statements, ETOP indicates design errors. The most common errors are duplicate names, which can result when more than one database is used to create a single dictionary design. At the end of the ETOP session, you will receive a message indicating duplicate file names.

Although ETOP can only flag duplicate file names, it does attempt to resolve duplicate element names. If two elements have the same name and identical attributes, including security, only one is included in the dictionary design. Because any element can be found in any number of files, this built-in mechanism for eliminating duplicate element definitions doesn’t make any difference to the resulting dictionary design. If, however, two elements have the same name but their attributes aren’t identical, ETOP adds both elements to the dictionary design and flags the second occurrence.

All errors must be corrected before PDL can create the dictionary. The pdlsrc.pdl file can be used as input to the PDL program.

Date Elements

ETOP does not use PowerHouse DATE datatypes in its generated PDL. To use DATE datatypes, you will have to modify the definition for Eloquence datatypes used as dates.

Refine Your Dictionary Design

You created the basic dictionary design with ETOP but you can enhance this skeletal design. The following list contains a sample of the types of changes you might want to make to the generated PDL:

- add indexed or flat files
- add item redefinitions
- add or change implied decimal positions in element types
- add element descriptions, headings, and labels
- add new logon IDs or UICs
- restrict access of application security class to files and elements

The ETOP Environment

At execution time, the run time environment has to be set up the same way as it is for all PowerHouse components (using, for example, the setpow.csh script on UNIX).

The PH_IMAGE_LIB environment variable must be set and must specify the name of the library containing the IMAGE emulator's version of the IMAGE intrinsics.

Because ETOP is built using the same standard environment as the other PowerHouse components, including our messaging system, it is possible to specify an alternative message file. The designated name for the ETOP message file is etopmsg.

If interrupted at any point by a user break action (<CTRL-C>), ETOP will issue the message "<User Break>" and will terminate.

Chapter 8: QCOBLIB Utility

Overview

This chapter documents how to use the utility QCOBLIB to generate COBOL definitions from a PDL dictionary.

Generating COBOL Definitions

QSHOW can be used to generate COBOL copy library definitions for the entities in your PDL dictionaries. To generate COBOL definitions, use the GENERATE statement. For example, to generate COBOL definitions for all the files in your dictionary, enter

```
> SET LANGUAGE COBOL
> GENERATE FILE ALL
```

To be more selective, you can use the following name-options:

- namelist
- PATTERN string
- SOUNDEX (string)

For information on these name-options, see (p. 134).

When you use the GENERATE FILE statement, a banner message for QCOBLIB appears. QCOBLIB is the PowerHouse utility that generates COBOL definitions. It can be run directly from the operating system by entering:

```
:QCOBLIB
```

However, when QCOBLIB is run from the operating system, it runs without prompting and creates COBOL definitions for all the files in your dictionary. If you don't want definitions for all your files, use the GENERATE statement.

The **parm=88** program parameter enables the creation of COBOL level-88 definitions from element values specifications in the dictionary. By default, level 88 definitions are not created.

The definitions created by the GENERATE statement are written to a file named QCOBTEXT. If QCOBTEXT already exists, a prompt is displayed asking you if it is okay to delete the existing QCOBTEXT. Enter Y (Yes) to proceed. If you reply N (No), the definitions are written to the temporary file, $OLDPASS. When COBOL definitions are successfully written to QCOBTEXT, a message is displayed:

```
The file QCOBTEXT has been created.
```

Two error messages may appear when you use the GENERATE statement.

```
Errors have occurred. Check QCOBLIST report.
Duplicate keys have occurred. Check QCOBLIST report.
```

The QCOBLIST report details all errors, as well as changes that have been made to the dictionary specifications. Changes to the dictionary specifications are made only when attempting to resolve incompatibilities between PowerHouse and COBOL.

By default, the QCOBLIST report is routed to the system printer. Any other printer can be specified by entering a file equation before running QCOBLIB, as in

```
:FILE QCOBLIST;DEV=LP2
```

Resolving Incompatibilities

When the GENERATE statement is used, an attempt is made to resolve the following incompatibilities between the dictionary and COBOL conventions.

File names

The dictionary file name, not the open name, is always used in the copy library.

Reserved words

If an entity name conflicts with any COBOL reserved word, the prefix "Q-" is added. For example, an element DIVISION becomes Q-DIVISION.

Illegal characters

Special characters not allowed in COBOL entity names are converted to hyphens. For example, P/O NUMBER becomes P-O-NUMBER.

Copy library key names

The first eight characters of the dictionary name are used to form the copy library key. Duplicate keys often result when long file names are truncated to eight characters. For example, the file names EMPLOYEEINDEX and EMPLOYEES produce the same copy library key, EMPLOYEE. The QCOBLIST report flags duplicate keys. All duplicate keys must be resolved before COBEDIT can be successfully run. Special characters found in the dictionary file name are changed to zeros in the copy library definition.

Signed items

An "S" is added to the copy library picture for a ZONED or PACKED item that is SIGNED, and for an INTEGER of SIZE 2, 4, or 8, regardless of whether they have a leading sign or trailing sign.

Incompatible datatypes

PowerHouse datatypes that are not compatible with HP COBOL types are converted as follows:

PowerHouse	COBOL Type
FREEFORM	X(n)*
FLOAT SIZE 4	X(4)
FLOAT SIZE 4	X(8)
PHDATE	X(2)
JDATE	X(2)
INTEGER SIZE 6	X(6)

* n is determined by the item size

Chapter 9: PH Integrator Utility

Overview

Note: **PowerHouse Integrator can be used with PHD dictionaries only.**

This chapter describes the PowerHouse Integrator utility. PowerHouse Integrator assists in the translation of CDD/Repository and PowerHouse dictionary.

This chapter is divided into the following parts:

- an overview of PowerHouse Integrator
- translating PowerHouse dictionaries to CDD/Repository
- translating CDD/Repository dictionaries to PowerHouse Dictionary Language (PDL)
- running PowerHouse Integrator
- designated files in PowerHouse Integrator
- PowerHouse Integrator program parameters
- a summary of PowerHouse Integrator statements
- the details of each PowerHouse Integrator statement

PowerHouse Integrator uses the syntax and language rules that are discussed in Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

What is PowerHouse Integrator?

CDD/Repository is an ORACLE data dictionary system that provides the ability to create, analyze, and administer metadata. CDD/Repository dictionaries created in CDD/Repository Dictionary Operator statements (CDO) can store not only definitions, but also information about how the definitions are related. You can create and manipulate definitions in CDO dictionaries through the CDO utility.

PowerHouse Integrator is a utility that translates CDD/Repository and PowerHouse data dictionaries to PDL and CDD dictionary languages. PowerHouse Integrator provides the basic framework from which you can begin coding the final dictionary. Your input may be necessary because of the different capabilities that exist between source and target dictionaries.

Familiarity with basic conversion concepts, advanced PowerHouse experience, and an in-depth knowledge of the applications being converted are necessary for successful translations.

What Does PowerHouse Integrator Translate?

PowerHouse Integrator translates CDD/Repository fields, records, RMS databases, and databases to equivalent PowerHouse Definition Language (PDL) statements. PowerHouse Integrator also translates PowerHouse dictionary elements, records, and files to equivalent CDO statements.

PowerHouse Integrator cannot translate certain information effectively. In some cases where a best try to translate is attempted, comments are issued in the generated code to show how the resulting syntax was obtained.

Note: A comment in CDO beings with an exclamation mark (!). In PDL, a comment begins with a semicolon (;).

The following table shows syntax that PowerHouse Integrator will attempt to translate with a best try:

Resulting CDD/Repository from PowerHouse
`EDIT_STRING`
`VALID IF <conditional-expression>`
`DATA TYPE determination`

Resulting PDL from CDD/Repository
`PICTURE/FORMAT/FLOAT,CHAR/LEADING SIGN, etc.(from EDIT_STRING)`
`DATA TYPE determination`

PowerHouse Integrator does not translate entities that it doesn't understand.

Syntax not Translated from CDD/Repository to PDL
`computed-by-clause`
`FILLER`
`INPUT_VALUE`
`JUSTIFIED`
`NAME FOR`
`VALID IF <expressions>`

Also, SELECT values on items are not translated into CDD/Repository from PowerHouse.

Note: Database statements for PowerHouse are translated into CDD/Repository comments in the same manner as the "FILE <filename> ORGANIZATION relational" syntax would be translated.

PowerHouse Integrator does not supply default attributes when translating from PowerHouse dictionary to PDL, or from a CDD/Repository dictionary to CDO. However, if the interface to the dictionary automatically supplies defaults for the missing attributes, the utility generates this information.

Security

PowerHouse Integrator only has access to dictionaries that you have access to. When PowerHouse Integrator accesses a PowerHouse dictionary, no security is enforced that limits access to entities or attributes. The security that the CDD/Repository access routines enforce is in effect when PowerHouse Integrator accesses a CDD/Repository dictionary.

Terminology Comparison Table

Term	PowerHouse Description	Term	CDD/Repository Description
Element	A logical data definition that specifies, for all like-named items, what data values are allowed and how they are to be reported or displayed; also specifies default storage attributes.	Field	A physical data definition that specifies exact storage attributes such as programming language and machine-specific datatype representation, storage size, and location within a record. Also specifies allowed values, reporting attributes and display attributes. Fields may be part of definitions of indexes and keys.

Terminology Comparison Table

Term	PowerHouse Description	Term	CDD/Repository Description
Item	A physical data definition that specifies storage attributes, such as programming language and machine-specific datatype representation, storage size, and location within a record. Items may be part of an index definition. An item inherits the editing and display characteristics of its like-named element.		
Record	An ordered list of related items which constitutes the basic unit of input/output. It includes related index and segment information.	Record	An ordered list of related fields or records which constitutes the basic unit of input/output.
File/Database	A physical "container" of records and items. Relates to a specific disk file, as specified by the Open name attribute. Specifies the method used to access the file (SYBASE for PowerHouse 4GL 8.30 only; ORACLE and ORACLE Rdb for all versions of PowerHouse 4GL) along with associated attributes (such as record format).	Database RMS_ DATABASE	A physical "container" of records and fields. It relates to a specific disk file. Specifies the RMS attributes for a file (such as record format, scope, indexes, etc.) and relates to one or more CDD/Repository databases.
Item Structure	An item which is subdivided into other items.	Structure	A field which is subdivided into other fields.
Coded Record	A non-relational file may contain several different record types, distinguished from one another by the value of a common item. These record types are known as "coded records".	Record Variant	A record may consist of a fixed portion and one or more variant portions. Each portion contains a set of fields, but the variant portions share the same part of the record; therefore, only one variant can be active at any given time, as determined by the value of a field in the fixed portion.

Terminology Comparison Table

Term	PowerHouse Description	Term	CDD/Repository Description
Redefined Item	A record may contain several different item structures which overlay the same area of the record, where a subsequent item redefines a previous item. For example, when two different items overlay the same area, the second item redefines the first one.		

Valid Entity Types

The following tables show what entity types are valid for translation and the results of translating them. Note that output depends on the source dictionary, which is either PowerHouse or CDD/Repository:

Valid Entities in the PowerHouse Source Dictionary

PowerHouse translatable entity type	PDL corresponding entity	CDD/Repository corresponding entity
element	referenced usage element	field(s)
record	all referenced usages all referenced elements record items indexes no file statement generated	all referenced fields record fields, structures and variants in the record
file	file with open name all records	RMS_DATABASE with record definition[1] all records defined index definitions

[1]In the case where more than one record is defined for a file, the record definition that PowerHouse Integrator creates during translation is the one described to the RMS database. PowerHouse Integrator generates the other record definitions, but they are not associated with the RMS database. Where multiple definitions exist for the same physical file and none is marked "CREATE" (for example, coded records), the definition that PHDFM creates is the first one translated.

Valid Entities in the CDD/Repository SOURCE DICTIONARY

CDD/Repository translatable entity type	CDD/Repository corresponding entity	PDL corresponding entity
field	field(s)	element

Valid Entities in the CDD/Repository SOURCE DICTIONARY

CDD/Repository translatable entity type	CDD/Repository corresponding entity	PDL corresponding entity
record	all referenced fields	all referenced usages all referenced elements
	record fields, structures and variants in the record	record items
		no file statement generated no indexes generated
RMS_DATABASE	RMS-DATABASE one record is defined index definitions	file with no open name record indexes
database	same as RMS_DATABASE	same as RMS_DATABASE open name now available for file

Translating PowerHouse Dictionary to CDD/Repository

Prior to starting the translation process, ensure that the source dictionary is accessible to PowerHouse Integrator. PowerHouse dictionary and application security is ignored when PowerHouse Integrator accesses it.

You can specify what entities you want to generate code for with the TRANSLATE statement. For more information, see the TRANSLATE statement (p. 218).

Element, Record, and Index Definitions

All element and RMS record definitions in a PowerHouse data dictionary can be translated into the appropriate CDD/Repository definitions. Index definitions are translated with the record. Where multiple definitions exist for the same physical file (for example, coded records), and one is marked for creating, the definition that PHDFM creates is the first one translated. You can choose the definition you want to use from those generated, or you can merge several of them into an appropriate variant structure.

In a PowerHouse dictionary, when an element has the same name as a record, PowerHouse Integrator will translate both entities. However, CDD/Repository will fail when the CDO utility is executed because record names are not allowed to have the same names as fields in CDO. In this situation, you can either change the names in PowerHouse dictionary before translation or edit the PowerHouse Integrator CDO output to avoid the naming conflict.

Missing and Default Information

PowerHouse dictionary provides default attributes for some elements and usages if you do not specify them. When you are translating a PowerHouse dictionary to CDO, PowerHouse Integrator supplies this default information to CDO.

Translating Elements and Items

The following table displays how PowerHouse dictionary elements and attributes are translated into CDD/Repository definitions:

PowerHouse Element/Item to CDD/Repository Field Translations

PowerHouse dictionary ELEMENT attribute	CDD/Repository FIELD statement
ELEMENT field-name	DEFINE FIELD field-name
DESCRIPTION "text"	DESCRIPTION [IS] /*text*/
USAGE IS name[1]	BASED ON name
DATATYPE type[2]	DATATYPE [IS] type
OUTPUT SCALE n	DISPLAY_SCALE [IS] n
PICTURE/FORMAT/SEPARATOR	[language] EDIT_STRING [IS] edit-string[3]
HELP string	HELP-TEXT IS string
INITIAL VALUE value[4]	INITIAL_VALUE [IS] value-expr
OCCURS n	OCCURS n [ITEMS]
HEADING string[5]	QUERY_HEADER [IS] string
VALUES n1 TO n2 [,n3 to n4]...	VALID IF conditional-expr
LABEL string	QUERY_NAME
BWZ	*
DECIMAL n	*
FILL char	*
FLOAT char	*
LEADING SIGN char	*
PATTERN pattern	*
SIGNIFICANCE n	*
TRAILING SIGN string	*

(*) no translation (may be included in the edit string).

[1]The BASED ON option in CDD/Repository does not operate in the same fashion as the USAGE option in PowerHouse dictionary. In CDD/Repository, one field is based on another field but any attribute of the BASED ON field may be overridden by specifying the option on the new FIELD statement. In PowerHouse, attributes set by the USAGE option can't be changed in the element definition.

[2]For a comparison of PHD and CDD/Repository datatypes, refer to the Datatypes table later in this section.

[3]An edit string is produced based on the element class and the information provided for such things as picture/format. You can change the edit string to take advantage of field clauses available in CDD/Repository.

[4]The initial value is a constant in PowerHouse dictionary and is translated directly to INITIAL for CDD/REPOSITORY.

[5]In CDD/Repository, the query_header may consist of more than one line. PowerHouse Integrator separates the lines by scanning the string from PowerHouse dictionary for the multi-line heading character. Each line of heading produces a query header string.

The following table shows PowerHouse element and item types and the corresponding CDD/Repository field types:

Corresponding Datatypes

PDL Element Type	PDL Item Type	CDD/Repository FieldType
character	character	text
	varying character	varying string
numeric	float size 4	F_FLOATING
	float size 8	D_FLOATING
	G_FLOAT	G_FLOATING
	integer size n	byte/word/longword, quadword,
	signed	octaword
	zoned	signed as appropriate
	signed	zoned
	unsigned	right overpunched
	numeric	unsigned numeric
	packed	zoned numeric
	freeform	packed decimal
	interval	leading separate
		signed quadword
date	PHDATE	word
	JDATE	word
	DATE (century included)	signed longword
	VMSDATE	date
	DATETIME	signed quadword[1]
	ZDATE	zoned numeric

[1]DATETIME is functionally two longwords, the first being the DATE, and the second being the TIME.

Item Attributes

The item datatype, size, and occurrence specifications are declared on the ITEM statement in PDL. However, CDD/Repository does not allow you to change the item datatype, size, or number of occurrences at the item level. To ensure the designer will find the problem, PowerHouse Integrator intentionally generates invalid CDO. In this case, create a new CDO field statement with the equivalent of the item attributes from the PowerHouse dictionary.

Resolving Conflict Syntax

When you translate the PowerHouse Dictionary to CDD/Repository, PowerHouse generates comments on invalid syntax in the following form:

!additional info: <name>TYPE<type>.

You can resolve conflicts of this nature by defining different fields for each distinct field type, and using them in the record definition.

Translating the SELECT Option

The SELECT option of the ITEM statement has no counterpart in CDD/Repository. As a result, when it encounters SELECT, PowerHouse Integrator generates a comment in the following form:

!additional info: SELECT VALUE <value>

To resolve this difference, you can either manually combine the source into multiple variants, or put the select value as a condition on the corresponding variant statement, or ignore the selection value.

Translating Record-structures

The following table lists the record-structure translation equivalents when translating record-structures from PowerHouse dictionary to CDD/Repository. Explanations of exceptions follow the table.

PowerHouse Dictionary to CDD/Repository Record Translation

PowerHouse Dictionary	**CDD/Repository**
RECORD record-name	DEFINE RECORD record-name
DESCRIPTION "text"	[DESCRIPTION [IS]/*TEXT*/
ITEM name[1]	name
STRUCTURE	[structure-name-clause]
CREATE/NOCREATE	no translation END [record-name-clause] RECORD

[1]Items may also be redefinitions of other items. Variants are the CDD/Repository equivalent of redefinition. For more information, see "Translating PowerHouse Dictionary Redefinitions", on (p. 192).

Translating BEGIN...END Structures

In PDL, BEGIN STRUCTURE and END STRUCTURE delineate an overlaid portion of the record buffer; however, in CDD/Repository variants are used to completely translate that area.

The following example shows PDL statements for an item structure translated into CDD/Repository statements.

PDL Redefinition	**CDD/Repository Variant Statement**
	VARIANTS. VARIANT. generated_structure_0 STRUCTURE.

ITEM name1	name1.
	END generated _structure_0
	STRUCTURE.
	END VARIANT.

BEGIN STRUCTURE	VARIANT. generated_structure_1 STRUCTURE.
ITEM name2	name2. END generated _structure_1 STRUCTURE.

END STRUCTURE	END VARIANT.
	END VARIANTS.

Translating PowerHouse Dictionary Redefinitions

In PDL, item redefinitions are another way to delineate an overlaid portion of a record buffer. The CDD/Repository equivalent is variants.

To ensure compatibility with programs such as DATATRIEVE and COBOL, PowerHouse Integrator generates code with VARIANT and STRUCTURE clauses when translating PowerHouse dictionary redefinitions, as shown in the following table:

PDL Redefinition	CDD/Repository Variant Statement
	VARIANTS.
	VARIANT.
	generated_structure_0 STRUCTURE.
ITEM name1	name1.
	END generated _structure_0
	STRUCTURE.
	END VARIANT.
	VARIANT.
	generated_structure_1 STRUCTURE.
ITEM name2	name2.
REDEFINES ITEM	
name1	END generated _structure_1
	STRUCTURE.
	END VARIANT.
	END VARIANTS.

Note that PowerHouse Integrator does not generate a CDD/Repository variant conditional expression which would be used to determine the proper variant structure to be referenced.

Translating Files

When you translate a file from PowerHouse dictionary to CDD/Repository, an RMS database definition is created. All of the associated record definitions are translated into CDD/Repository record definitions. The RMS database definition includes the record and its associated index definitions. Since the file open name is not part of the RMS database, this information is retained only in a comment line in the translated source.

CDD/Repository uses the DEFINE DATABASE statement to create the file. PowerHouse Integrator places the statement in a comment line to ensure that it is not executed.

When a single PowerHouse dictionary file that is to be translated into CDD/Repository has many record definitions using the coded record concept, PowerHouse Integrator does the following:

- It generates an RMS database definition based on the PowerHouse file information.
- It generates all the record definitions for the file in CDO.
- It associates the record marked with the CREATE option in PowerHouse with the RMS database definition in CDO.

To create the equivalent of a PowerHouse dictionary coded record-structure, conditions must be placed on the CDO variant clauses. PowerHouse Integrator does not generate the CDD/Repository record definition with all the variant clauses in place because PowerHouse Integrator cannot determine the correct record definition (what fields belong in the variants, which belong outside of the variants, and so on). In addition, CDD/Repository cannot accept a segment reference to a field in a variant portion in an index definition.

Translating Indexes

In PowerHouse dictionary, indexes are associated with the record, not with the file. In CDD/Repository, indexes (called keys) are associated with the RMS database, which CDD$DATABASE associates with a physical file. An RMS database may be used by more than one database. An RMS database can only contain one record definition, but a record may be used by more than one RMS database.

Note that indexes that belong to an RMS database can only reference items that are not part of a variant structure. This means that some index segments in a PowerHouse dictionary may not translate because they are substructures of another item.

The following table shows index to key translation:

PDL Statements	**CDD/Repository Statements**
	KEYS
INDEX index_name_1 [REPEATING] [NULL] [NULL_VALUE null value] SEGMENT field-name1 SEGMENT field-name2	KEY 0 [DUPLICATES] [NULL_KEY] [NULL_VALUE null value] field-name1 IN record-name field-name2 IN record-name
INDEX index_name_2 [REPEATING] [NULL] [NULL_VALUE null value] SEGMENT field-name1 SEGMENT field-name2	KEY 1 [DUPLICATES] [NULL_KEY] [NULL_VALUE null value] field-name1 IN record-name field-name2 IN record-name ... . END. END record-name RMS_DATABASE.

Output

The output of a PowerHouse dictionary to CDD/Repository translation is CDD/Repository source that can be used by CDO. PowerHouse Integrator supplies the extension .CDO to the output file.

The output file begins with comments stating the source dictionary name, the user ID of the translator, and the date and time that translation began.

The first source line in the file is SET VERIFY - the CDD/Repository statement that echoes the statements and comments before they are executed. If no errors are encountered, PowerHouse Integrator displays the PowerHouse prompt.

Additional status messages are sent to the terminal in interactive session and to the job listing file in batch session (SYS$OUTPUT).

Translating CDD/Repository to PDL

Prior to starting the translation process, ensure that the source dictionary is accessible to PowerHouse Integrator. CDD/Repository dictionary security is enforced as PowerHouse Integrator reads the entities in the CDD/Repository dictionary.

You can specify what entities you want to generate code for with the TRANSLATE statement. For more information, see the TRANSLATE statement (p. 218).

Field, RMS Record, and Index Definitions

All field and RMS record definitions can be translated into the appropriate PDL structures. Index definitions can also be translated.

Missing and Default Information

PowerHouse Integrator translates the available entities and attributes in the source dictionary. Information about dictionary attributes is not supplied by the utility because CDD/Repository doesn’t supply default or missing information.

Translating Fields

The following table shows how CDD/Repository fields are translated into ELEMENT statements:

CDD/Repository Field Statement	PDL Element Statement
`DEFINE FIELD field-name`	`ELEMENT element-name`
`DESCRIPTION [IS]/*text*/`	`DESCRIPTION "text"`
`AUDIT [IS]/*text*`	`no translation`
`array-type`[1]	`limited translation`
`BASED ON field-name`[2]	
`computed-by-clause`	`no translation`
`DATATYPE [IS] data-type`[3]	
`EDIT_STRING [IS] edit-string`[4]	`limited translation`
`FILLER`	`no translation`
`HELP_TEXT IS quoted-string`	`HELP quoted_string`
`INITIAL_VALUE [IS] value_expression`[5]	`INITIAL VALUE value expression`
`INPUT-VALUE {REQUIRED OPTIONAL}`	`no translation`
`JUSTIFIED justified-optional`	`no translation`
`NAME [FOR] language [IS] name`	`no translation`
`OCCURS n [TIMES]`	`OCCURS n`
`QUERY_HEADER [IS] quoted-string [,quoted-string]...`[6]	`HEADING quoted-string`
`QUERY_NAME [IS] {quoted-string/query-name}`	`LABEL`
`VALID IF conditional-expression`	`no translation`

[1]It's possible to define a multi-dimensional array with specified bounds in CDD/Repository. However, PowerHouse only supports a one-dimensional array with implicit bounds of 1 and n. For the simple case of a one-dimensional array with bounds of 1 and n, the proper occurs clause is generated. For additional dimensions or bounds, a character element or item of the proper size is generated. This precaution is taken to prohibit subscript references that are no longer in the bounds of the array or that reference the incorrect occurrence.

[2]The BASED ON option is not the same as the USAGE option in the PowerHouse dictionary. In CDD/Repository, one field is based on another field, but any attribute of the based on field can be overridden by specifying the option on the new FIELD statement. In a PowerHouse dictionary, certain attributes set by a usage can't be changed in the element definition.
When PowerHouse Integrator encounters a field that is based on a previous field, it will generate both a USAGE and an ELEMENT statement for the previous field. The designer should verify that these statements are appropriate for his dictionary.

[3]For a comparison of CDD/Repository and PHD datatypes, see the next table.

[4]For numeric fields, a picture is created. For date fields, a format and separator are created.

[5]The INITIAL_VALUE can be an initial expression, but PowerHouse Integrator only translates constant values.

[6]In PDL, the heading can be up to 60 characters long. Multiple lines are indicated by the multi-line heading character, which is a caret (^). In CDD/Repository, the query_heading may consist of more than one line. When translating the CDD/Repository into PDL, each line is concatenated together, separated by a caret (^). There is no check to verify that the total size is less than or equal to 60 characters. PDL issues an error message if the lines in the header exceed 60 characters.

The following table shows CDD/Repository Datatypes and the corresponding PDL element and item types.

CDD/Repository Datatypes	PDL Element Types	PDL Item Types
ALPHABETIC [SIZE IS] n [CHARACTERS]	CHARACTER SIZE n	CHARACTER
BIT [SIZE IS] n[1]	CHARACTER SIZE (n+7)/8	CHARACTER
DATE	DATE	VMSDATE
decimal string types[2]		
fixed point types[3]		
floating point types[4]		
POINTER [TO name [IN name]...]	NUMERIC	INTEGER SIZE 4
REAL	NUMERIC	FLOAT SIZE 4
SEGMENTED STRING [SEGMENTED_LENGTH [IS] n BYTES] [SEGMENTED_TYPE [IS] m]	not supported	not supported
TEXT [SIZE IS] n [CHARACTERS]	CHARACTER SIZE n	CHARACTER SIZE n
UNSPECIFIED [SIZE IS] n BYTES	CHARACTER SIZE n	CHARACTER SIZE n
VARYING_STRING [SIZE IS] n [CHARACTERS]	CHARACTER SIZE n	VARCHAR SIZE n

1 See the following section, "BIT".
2 See "Decimal String" on (p. 196).
3 See "Fixed Point" on (p. 197).
4 See "Floating Point" on (p. 197).

BIT

PowerHouse Integrator assumes that BIT fields have extra bits inserted so that any subsequent fields are byte-aligned. This means that BIT SIZE 3 and BIT SIZE 6 would each occupy one byte; a BIT SIZE 2 followed by a BIT SIZE 5 would occupy two bytes (one byte for each field).

The following options, available only on the CDD/Repository alphabetic, text, and varying string datatypes, map to the following element options in the PowerHouse dictionary:

LOWERCASE	maps to	DOWNSHIFT
UPPERCASE	maps to	UPSHIFT
CASE_INSENSITIVE	maps to	nothing_specified

Decimal String

The translations of decimal string types are straightforward. The size becomes the element size and the storage size is calculated appropriately. Scale becomes the input scale for the element.

Decimal String Types

CDD/Repository	PDL
LEFT OVERPUNCHED NUMERIC	not supported
LEFT SEPARATE NUMERIC	not supported

Decimal String Types

CDD/Repository	PDL
RIGHT SEPARATE NUMERIC	not supported
RIGHT OVERPUNCHED NUMERIC	ZONED SIGNED
PACKED DECIMAL	PACKED
ZONED NUMERIC	ZONED NUMERIC
UNSIGNED NUMERIC	UNSIGNED ZONED

All of the above CDD/Repository decimal string types can be followed by:

[[SIZE IS] m [DIGITS]] [SCALE n]]

where n is between -128 and 127 for CDD/Repository, between -16 and 16 for PDL, and m is between 0 and 32 for both CDD/Repository and PDL.

Fixed Point

The CDD/Repository syntax for declaring fixed point datatypes is

[SIGNED|UNSIGNED]
{BYTE|WORD|LONGWORD|QUADWORD|OCTAWORD}

[[SIZE IS] m [DIGITS] [SCALE n]]

where n is between -128 and 127 for CDD/Repository, between -16 and 16 for PDL, and m is between 0 and 32 for both CDD/Repository and PDL.

Fixed Point Item Sizes

SIZE (Bytes)	CDD/Repository	PDL
1	BYTE	INTEGER SIZE 1
2	WORD	INTEGER SIZE 2
4	LONGWORD	INTEGER SIZE 4
8	QUADWORD	INTEGER SIZE 8
16	OCTAWORD	not supported

PowerHouse does not support the OCTAWORD datatype. Size, if specified, is the element size. The storage size for the integer item is determined by the original datatype. Scale becomes the input scale for the element.

Floating Point

The CDD/Repository syntax for declaring floating point datatypes is:

{D_FLOATING|F_FLOATING|G_FLOATING|
H_FLOATING}[COMPLEX]SCALE n]

where n is between -128 and 127 for CDD/Repository and -16 and 16 for PDL.

In PowerHouse, H_FLOAT and complex values are not supported. In all other cases, the item becomes a float of the proper size.

CDD/Repository	PDL
D_FLOAT	FLOAT SIZE 8
F_FLOAT	FLOAT SIZE 4

CDD/Repository	PDL
G_FLOAT	G_FLOAT

Translating Record-structures

The following table shows how CDD/Repository record-structures are translated into PDL statements:

Corresponding Record Attributes	
CDD/Repository statement	**PDL statement**
DEFINE RECORD record-name	RECORD record-name
[DESCRIPTION [IS]/*text*/]	DESCRIPTION " text "
[AUDIT [IS]/*text*/]	no translation
[array-type]	no translation
[NAME FOR language [IS] name]	no translation
[occurs-depending-clause]	no translation
include-name-clause	refer to description
[aligned-clause]	no translation
[structure-name-clause]	refer to description
[variant-clause]	refer to description
END [record-name] RECORD	no translation

include-name-clause

In CDD/Repository, the name can be a field or record definition. CDD/Repository names are translated to PDL item names, while CDD/Repository record definitions are expanded to show their contents when they are translated to PDL.

As a result of translation from CDD/Repository to PDL, the looping of record-structures can occur. For example, record A can be included in record B and record B can be included in record A. While this is possible in CDD/Repository, PDL will give error messages if you attempt to compile the statements that PowerHouse Integrator generates in this situation.

structure-name-clause

A PowerHouse substructure, like a CDD/Repository variant, provides a convenient method of grouping a collection of related items into a single meaningful unit. An item that has a substructure can itself be part of a larger substructure.

The following table shows what PDL statements the CDD/REPOSITORYSTRUCTURE clause is translated to:

CDD/Repository Statement	PDL Statement
structure-name STRUCTURE	ITEM structure-name[1] BEGIN STRUCTURE
[DESCRIPTION [IS]/*text*/]	comments are generated
[record-attributes]	no translation
[OCCURS]	PUT OCCURS on ITEM statement above
[aligned clause]	no translation
[include-name-clause]	refer to description

CDD/Repository Statement	PDL Statement
[structure-name-clause]	refer to description
[variant-clause]	refer to description
END [structure-name] STRUCTURE.	END STRUCTURE

[1]A new item is created to correspond to the CDD/Repository structure clause because there are a few cases where PDL does not allow a structure but CDD/Repository does.

In CDD/Repository, you can define a structure that repeats based on the content of a numeric field in the same record by using the OCCURS DEPENDING ON clause. The PDL item that is generated for the structure is generated so that it repeats the maximum number of times.

When a structure is encountered as the first item in a CDD/Repository record, an item definition is generated in PDL. What follows is an example of substructures:

Structure as the First Part of a Previous Structure	
CDD/Repository syntax	**PDL syntax**
A STRUCTURE.	ITEM A CHARACTER SIZE n BEGIN STRUCTURE
B STRUCTURE.	ITEM B CHARACTER SIZE m BEGIN STRUCTURE
B1. B2.	ITEM B1 ITEM B2
END B STRUCTURE.	END STRUCTURE
A1. A2.	ITEM A1 ITEM A2
END STRUCTURE.	END STRUCTURE

The PDL ITEM statements for A and B correspond to the CDD/Repository STRUCTURE statements for A and B. The size of each of these items is equal to the size of the item's substructure. PDL requires the second item B since it does not allow a structure to be the first item of another structure.

variant-clause

In CDD/Repository, a record may consist of both fixed and variant sets of fields. The variant portions overlay each other; therefore, only one variant can be active at a time. This is determined by the value of the field in the fixed portion of the record.

The following is the general form of the VARIANT clause mapped against PDL syntax.

CDD/Repository statements	PDL statements
VARIANTS.	ITEM generated_element_n0
VARIANT.	ITEM generate_element_n1 & REDEFINES generated_element_n0 BEGIN STRUCTURE
[EXPRESSION [IS] conditional expression].	no translation
[included-name-clause]...	
[structure-name-clause]... [variant-clause]...	
END VARIANT.	END STRUCTURE

CDD/Repository statements	**PDL statements**

`VARIANT.`	`ITEM generated_eleemnt_n2 &`
	`   REDEFINES ITEMS`
	`generated_element_n0`
	`BEGIN STRUCTURE`

`   [EXPRESSION [IS]`
`conditional-expression].`
	`no translation`

`[included-name-clause]...`	

`[structure-clause]...`
`       [variant-clause]...`	

`END VARIANT.`	`END STRUCTURE`
`.` `.` `.`	
`END VARIANTS.`	`no translation`

The table above shows that with variant structures, PowerHouse Integrator generates an element whose size is the size of the longest variant structure.

This item serves as the base for all the subsequent translations of the variant structures. For each variant, it generates an item whose size is the size of the variant. These items redefine the base item. For each of these items an item structure is generated that contains each of the fields, structures, and variants in the corresponding variant.

In the following examples of variants and substructure combinations, PowerHouse Integrator generates pseudo-elements since PDL does not allow a structure to be the first item of another structure.

In the table following, a variant containing the fields B1 and B2 is the first part of structure A, and shares the same space within the structure as the variant containing field C. Fields A1 and A2 comprise the last part of structure A.

A Variant as the First Part of a Previous Structure	
CDD/Repository syntax	**PDL syntax**

`A STRUCTURE.`	`ITEM A`
	`BEGIN STRUCTURE`

`   VARIANTS.`	`   ITEM generated_element_0`
	`   ITEM generated_element_1 &`
	`      REDEFINES generated_element_0`

`      VARIANT.`	`   BEGIN STRUCTURE`

`         B1.`
`         B2.`
	`      ITEM B1`
	`      ITEM B2`

`      END VARIANT.`	`   END STRUCTURE`

`      VARIANT.`	`   ITEM generated_element_2 &`
	`      REDEFINES generated_element_0`
	`   BEGIN STRUCTURE`

`        C.`	`      ITEM C`
`      END VARIANTS.`	`   END STRUCTURE`

`      A1`
`      A2.`
	`   ITEM A1`
	`   ITEM A2`

`END A STRUCTURE.`	`END STRUCTURE`

PowerHouse Integrator generates an item A, corresponding to the CDD/Repository structure A. The items GENERATED_ELEMENT_0, GENERATED_ELEMENT_1, and GENERATED_ELEMENT_2 are created as part of the translation of the variant clauses. Item A is substructured by items GENERATED_ELEMENT_0, A1, and A2.

Item GENERATED_ELEMENT_1 redefines GENERATED_ELEMENT_0 and is substructured by items B1 and B2, which correspond to the first variant. GENERATED_ELEMENT_0 is also redefined by item GENERATED_ELEMENT_2, which is substructured by item C, corresponding to the second variant. The last part of item A is comprised of items A1 and A2.

In PDL, the equivalent of a structure as the first part of a variant is not allowed, so PowerHouse Integrator makes the following conversion:

A Structure as the First Part of a Variant

CDD/Repository statements	PDL statements
`VARIANTS.`	`ITEM generated_element_0`
`    VARIANT.`	`ITEM generated_element_1 &`
	`   REDEFINES generated_element_0`
	`BEGIN STRUCTURE`
`      A STRUCTURE.`	`   ITEM A`
	`   BEGIN STRUCTURE`
`         A1.`	`      ITEM A1`
`         A2.`	`      ITEM A2`
`      END A STRUCTURE.`	`   END STRUCTURE`
`      B1.`	`   ITEM B1`
`      B2.`	`   ITEM B2`
`   END VARIANT.`	`   END STRUCTURE`
`   VARIANT.`	`   ITEM generated_element_2 &`
	`      REDEFINES generated_element_0`
	`   BEGIN STRUCTURE`
`        C.`	`      ITEM C`
`   END VARIANT.`	`   END STRUCTURE`
`END VARIANTS.`	

GENERATED_ELEMENT_1 and GENERATED_ELEMENT_2 are created to form the equivalent of the two variant structures. GENERATED_ELEMENT_1 contains the structure A, which corresponds to the CDD/Repository structure A, and the fields A1, A2, B1, and B2. An element statement is created for item A. GENERATED_ELEMENT_2 contains the field C and redefines the first structure. The item GENERATED_ELEMENT_2 is required because PDL does not allow a substructure to be the first part of another structure.

In the following example, the first large variant is comprised of two other smaller variants and two fields, B1 and B2; the first small variant is composed of the fields Z1 and Z2, and is overlaid by the second small variant, which is composed of field X. The entire large variant is overlaid by another variant, containing field C.

A Variant as the First Part of a Variant

CDD/Repository syntax	PDL syntax
`VARIANTS.`	`ITEM generated_element_0`
`    VARIANT.`	`ITEM generated_element_1 &`
	`   REDEFINES generated_element_0`
	`BEGIN STRUCTURE`
`      VARIANTS.`	`   ITEM generated_element_2 &`
`         VARIANT.`	`   ITEM generated_element_3 &`
	`      REDEFINES generated_element_2`
	`   BEGIN STRUCTURE`
`            Z1.`	`      ITEM Z1`
`            Z2.`	`      ITEM Z2`
`         END VARIANT.`	`   END STRUCTURE`

A Variant as the First Part of a Variant	
CDD/Repository syntax	**PDL syntax**
`VARIANT.`	`ITEM generated_element_4 & REDEFINES generated_element_2 BEGIN STRUCTURE`
`X.`	`ITEM X`
`END VARIANT.`	`END STRUCTURE`
`END VARIANTS.`	
`B1.` `B2.`	`ITEM B1` `ITEM B2`
`END VARIANT.`	`END STRUCTURE`
`VARIANT.`	`ITEM generated_element_5 & REDEFINES generated_element_0 BEGIN STRUCTURE`
`C.`	`ITEM C`
`END VARIANT.` `END VARIANTS.`	`END STRUCTURE`

PowerHouse Integrator generates an item, GENERATED_ELEMENT_1, corresponding to the first large variant. A second generated item, GENERATED_ELEMENT_3, which corresponds to the first small variant, begins the initial structure, and is itself substructured by items Z1 and Z2. A third generated item, GENERATED_ELEMENT_4 redefines GENERATED_ELEMENT_2, and contains the item X, corresponding to the second small variant. Items B1 and B2 complete the initial structure, which is redefined by GENERATED_ELEMENT_5, (which corresponds to the final variant), which contains a structure that includes item C.

Translating RMS Databases

The RMS database defines a physical file description (but does not define an actual physical file). It may only have one record description associated with it. Therefore, translating an RMS database results in a PDL FILE statement (without a supplied open name), the associated PDL file description, and the associated PDL indexes described in "Translating Indexes" on (p. 203).

CDD/Repository syntax	**PDL syntax**
`DEFINE RMS_DATABASE <db-name>`	`FILE <db-name>`
`DESCRIPTION`	`DESCRIPTION`
`AUDIT`	`no translation`
`RECORD <name>`	`RECORD <name>`
`file-definition`	`defines ORGANIZATION for file`
`area-definition`	`no translation`
`keys-definition`	`see "Translating Indexes" on (p. 203)`
`END <db-name>`	`no translation`

File Statement Required in PDL

When translating from CDD/Repository to PDL, a file statement must be created. The RMS database name is used as the file name. The file open name can't be retrieved from information specified on the DEFINE DATABASE statement because the RMS database could be used for more than one DEFINE DATABASE. As a result, the open name is not supplied.

Translating Databases

Translating the entity that is created by the "define database" statement is essentially the same as translating an RMS database (since the defined database is a disk file with the specified attributes on an RMS database), except that the file open name is now available for use. Descriptions associated with the database become associated with the file in PDL.

Translating Indexes

In CDD/Repository, indexes are associated with the RMS database which can only contain one record definition. However, in PowerHouse, a file can be associated with several records, each of which may have different indexes. The following table shows how the index information in the CDD/Repository RMS database definition translates into PDL INDEX and SEGMENT statements that would follow the associated RECORD statements.

Key to Index Translation

CDD/Repository statements	PDL statements
`KEYS.`	`INDEX rms_database_name_0 &`
`   KEY 0`	`   [REPEATING]`
`      [DUPLICATES]`	`   [NULL]`
`      [NULL_KEY]`	`   [NULL_VALUE null-value]`
`      [NULL_VALUE null-value]`	`   SEGMENT field-name1`
`      field-name1 IN record-name`	`   SEGMENT field-name2`
`      field-name2 IN record-name`	
`   KEY 1`	`INDEX rms_database_name_1 &`
`      [DUPLICATES]`	`   [REPEATING]`
`      [NULL_KEY]`	`   [NULL]`
`      [NULL_VALUE null-value]`	`   [NULL_VALUE null-value]`
`      field-name1 IN record-name`	`   SEGMENT field-name1`
`      field-name2 IN record-name`	`   SEGMENT field-name2`
`      ...`	`   ...`
`      .`	
`   END.`	
`END  <record-name> RMS_DATABASE.`	

Output

The output of CDD/Repository to PDL translation is PDL statements that can be used to load a PowerHouse dictionary. PowerHouse Integrator supplies the extension .PDL to the output file.

The output file begins with comments stating the source dictionary name, the user ID of the translator, and the date and time that translation began.

Additional status messages are sent to the terminal in interactive session and to the job listing file in batch session (SYS$OUTPUT).

An Example of Substructuring

This example shows sample PDL and CDO code for a record that contains a redefinition of an item. The redefinition contains a substructure. The QSHOW output following the sample code shows the actual offsets of each item within the record-structure.

PDL Code

```
>   RECORD INVOICE_MASTER
>   ITEM INVOICE_DATE DATATYPE ZONED SIZE 8
>   ITEM INV_DT_PARTS DATATYPE ZONED SIZE 8 &
>      REDEFINES INVOICE_DATE
>      BEGIN STRUCTURE
>         ITEM INV_DT_YEARDATATYPE ZONED SIZE 4
>         ITEM INV_DT_MONDATATYPE ZONED SIZE 2
>         ITEM INV_DT_DAYDATATYPE ZONED SIZE 2
>   END STRUCTURE
.
.
.
```

Note: Item INV_DT_PARTS redefines item INVOICE_DATE, and is substructured by INV_DT_YEAR, INV_DT_MON, and INV_DT_DAY.

CDO Code

```
CDO> DEFINE RECORD INVOICE_MASTER.
CDO>  VARIANTS.
CDO>   VARIANT.
CDO>      INV_DT STRUCTURE.
CDO>          INVOICE_DATE TYPE UNSIGNED NUMERIC SIZE IS 8.
CDO>      END INV_DT STRUCTURE.
CDO>   END VARIANT.
CDO>   VARIANT.
CDO>      INV_DT_PARTS_1 STRUCTURE.
CDO>          VARIANTS.
CDO>            VARIANT.
CDO>              INVDT_PARTS_2 STRUCTURE.
CDO>                INV_DT_PARTS.
CDO>              END INV_DT_PARTS_2 STRUCTURE.
CDO>          END VARIANT.
CDO>          VARIANT
CDO>            INV_DT_PARTS_3 STRUCTURE.
CDO>              INV_DT_YR TYPE UNSIGNED NUMERIC SIZE IS 4.
CDO>              INV_DT_MON TYPE UNSIGNED NUMERIC SIZE IS 2.
CDO>              INV_DT_MON TYPE UNSIGNED NUMERIC SIZE IS 2.
CDO>            END INV_DT_PARTS_3 STRUCTURE.
CDO>        END VARIANT.
CDO>      END VARIANTS.
CDO>   END INV_DT_PARTS_1 STRUCTURE.
CDO>  END VARIANT.
CDO> END VARIANTS.
.
.
.
```

QSHOW Output Showing Items Offsets Within the Record-structure

The QSHOW output shows the actual offsets of each item within the record-structure. Since INV_DT_PARTS redefines INVOICE_DATE, they both have the same offset within the record. Similarly, since INV_DT_YR, INV_DT_MON and INV_DT_DAY are part of a substructure, they overlay INV_DT_PARTS within the record.

```
1991/09/12              SALES                                Page 1

                        R E C O R D  R E P O R
                        T

       For DICTIONARY:  PATH$FINANCE: [FINANCE]SALES.PHD;
Record:                 INVOICE_MASTER
of File:                INVOICE_MASTER
Organization:           INDEXED
Type:                   RMS
Open:                   INVOMAS
Record Format:          Fixed
Supersede:              No
Record Size:            64 Bytes
```

```
-- Record Contents --

     Item                Type                    Size        Offset
     INVOICE DATE        ZONED UNSIGNED          Occ              0
    _INV_DT_PARTS        ZONED UNSIGNED              8            0
     .INV_DT_YR          ZONED UNSIGNED              8            0
     .INV_DT_MON         ZONED UNSIGNED              4            4
     .INV_DT_Day         ZONED UNSIGNED              2            6
     .                                               2
     .
     .
```

Running PowerHouse Integrator

There are two ways to run PowerHouse Integrator:

- enter

  ```
  $ PHINTEGRATOR
  ```

  at the operating system prompt (set up the PowerHouse environment first).
- enter

  ```
  > $ PHINTEGRATOR
  ```

  and the program parameters from within another PowerHouse component.

Exiting PowerHouse Integrator

To return to the invoking program when you finish a PowerHouse Integrator session, enter the EXIT statement.

Designated Files in PowerHouse Integrator

The designated files used by PowerHouse Integrator for its internal use are listed below:

File	Extension	Purpose
PHINTMSG	.TXT	Optional message file that, if present, contains messages to be used by PowerHouse Integrator.
PHINTSAVE	.DTU	Temporary save file to which entered PowerHouse Integrator statements are written.
PHINTUSE	.DTU	Optional use file processed automatically by PowerHouse Integrator before opening the system input file (unless the auto program parameter has been specified).

PowerHouse and CDD/Repository Datatypes

For information regarding the relationship between PowerHouse datatypes and CDD/Repository datatypes, please see the section, "Relational PowerHouse Datatypes", in Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

PowerHouse Integrator Program Parameters

The PowerHouse Integrator program parameters are as follows:

Program Parameter	Description
auto	Establishes the file, screen, or report that is processed when the component is initiated.
dcl/nodcl	Stipulates whether or not operating system commands can be entered or executed when PowerHouse encounters an operating system prompt.
list|nolist	Establishes whether or not the PowerHouse component displays the source statement file.
osaccess/noosaccess	Stipulates whether or not operating system commands can be entered or executed when PowerHouse encounters an operating system prompt.
procloc	Causes PowerHouse to search for process files in a location other than the current directory.
prompt	Specifies the prompt for the PowerHouse component.
trusted/notrusted	Activates or deactivates C2-level security for the execution of RUN commands and DCL commands within components.
version	Provides the build number of the PowerHouse version.

Summary of PowerHouse Integrator Statements

The following table summarizes the purpose of each PowerHouse Integrator statement:

Statement	Purpose
EXIT	Ends a PowerHouse Integrator session.
GENERATE	Specifies what type of dictionary the information is targeted for.
GO	Begins execution of the request that has been entered.
QSHOW	Runs QSHOW.
REVISE	Edits the current temporary save file or specified file.
SAVE	Saves the PowerHouse Integrator source statements in a file.
SOURCE	Specifies the dictionary name and type being used.
TRANSLATE	Specifies the entities to be translated.

Statement	Purpose
USE	Processes PowerHouse Integrator source statements that are contained in a file.

EXIT

Ends a PowerHouse Integrator session.

Syntax

EXIT

Discussion

The EXIT statement ends the PowerHouse Integrator session and returns control to the operating system or to the invoking program.

GENERATE

Specifies the type of dictionary the information is targeted for.

Syntax

GENERATE [PDL|CDO] TO filespec

PDL|CDO

Generates PDL or CDO definitions for all the entities in the data dictionary.

PDL

Specifies that the target language for the translation is the PowerHouse Data Definition Language.

CDO

Specifies that the target language for the translation is Oracle's OpenVMS CDD/Plus Dictionary Operator statements.

Default: CDO

TO filespec

Specifies the file to which the output of the translation is sent.

A filespec is a name of an OpenVMS file (which may consist of the node, device, directory, filename, type, and version) or a logical name. The maximum length for a filespec in PowerHouse for OpenVMS is 255 characters. Filespecs are restricted to alphanumeric and punctuation characters. The characters ; $ and leading question mark (?) have special meanings in PowerHouse and are prohibited. A file specification takes the general form:

[NODE::][DEVICE:][[DIRECTORY]]FILENAME.EXT;1

The square brackets are required when you enter a directory name.

Discussion

The GENERATE statement defines the target language and the file specification for the output text file. This file can be specified with a logical name.

A default extension of .CDO or .PDL is applied to the file if no file extension is specified by the user.

At least one GENERATE statement is required, and although multiple GENERATE statements are allowed, the latest one specified prior to the GO statements is the option that is used. Multiple GENERATE statements do not alter the restrictions defined in the TRANSLATE statement.

The output file is opened when all information has been successfully loaded. CDD/Plus will be generated by default.

If a file with the specified name already exists, you are warned and given the option of replacing it or creating a new file. Subsequent GENERATE statements cause PowerHouse Integrator to append statements to the file.

Examples

The following example specifies that CDO code is to be generated.

```
> GENERATE TO CDO_OUTPUT_FILE
```

Generating CDD Definitions

To create a source file named CDO_OUTPUT_FILE.CDO, enter

```
> GENERATE CDO TO CDO_OUTPUT_FILE
```

The file CDO_OUTPUT_FILE.CDO will contain the CDD definitions.

Loading a New PowerHouse Dictionary with Generated Definitions

The GENERATE statement can be used to load a new PowerHouse dictionary with definitions from a CDD/Plus or PowerHouse dictionary. This eliminates the need to re-enter definitions. System options will always be generated. The following example shows the statements entered, using PowerHouse Integrator and PHDPDL, to load a new dictionary with generated definitions:

In PowerHouse Integrator,

```
> SOURCE DICTIONARY OLD TYPE PHD
> TRANSLATE ALL
> GENERATE PDL TO PDL_OUTPUT_FILE
> GO
```

creates a source file named PDL_OUTPUT_FILE.PDL.

In the following example in PHDPDL, the dictionary NEW is created using the CREATE DICTIONARY statement. The USE statement processes the definitions in the file PDL_OUTPUT_FILE.PDL. The LOAD statement enters the definitions into the NEW dictionary.

```
> CREATE DICTIONARY NEW NOT PRELOADED
> USE PDL_OUTPUT_FILE
> LOAD
> EXIT
```

Use the NOT PRELOADED option of the CREATE DICTIONARY statement when loading definitions from an old dictionary that does not contain the standard usages. If source code is generated for the entire dictionary, the usages from the old dictionary are loaded into the new one.

GO

Begins execution of the request that has been entered.

Syntax

```
GO
```

Discussion

The GO statement starts the translation process. You may repeat it as often as desired. If a GO statement is followed by a GENERATE statement and another GO statement, the second GO statement will simply generate the same set of entities and attributes as the previous GO statement, except for the new output field and target language.

If you press [Ctrl-C], the system will ask you if you want to continue. If you enter Yes, processing will continue. If you enter No, processing will stop and the output file will close if it was open.

If there is a file with the same name as the output file you are trying to create, a message is displayed and you are asked whether a new version of the file is to be created or not. When the translation is complete, the output file is closed and the prompt character (>) is displayed.

QSHOW

Runs QSHOW.

Syntax

QSHOW

Discussion

The QSHOW statement initiates a QSHOW session. QSHOW enables you to make quick on-line inquiries about entities (such as elements, files, and record-structures) in the data dictionary.

QSHOW is ready when its prompt string (QSHOW>) appears.

When you exit from QSHOW, your session resumes at the point at which it was interrupted.

For more information about QSHOW, see Chapter 4, "QSHOW Statements", in the *PDL and Utilities Reference* book.

REVISE

Edits the current temporary save file or a specified file.

Syntax

REVISE {* |filespec} [option]

*

Indicates that the current PHINTSAVE file is to be revised. If you specify options without specifying the name of a screen, you must use the asterisk.

filespec

Specifies the name of a file. This is the option to use when you want to revise a file other than the PHINTSAVE file.

A filespec is a name of an OpenVMS file (which may consist of the node, device, directory, filename, type, and version) or a logical name. The maximum length for a filespec in PowerHouse for OpenVMS is 255 characters. Filespecs are restricted to alphanumeric and punctuation characters. The characters ; $ and leading question mark (?) have special meanings in PowerHouse and are prohibited. A file specification takes the general form:

[NODE::][DEVICE:][[DIRECTORY]]FILENAME.EXT;1

The square brackets are required when you enter a directory name.

Options

The options are DETAIL, NODETAIL, LIST, NOLIST, USE, and NOUSE.

DETAIL|NODETAIL

DETAIL copies the contents of the revised file into the PHINTSAVE file after the revision is completed; NODETAIL doesn't.

If a file other than PHINTSAVE is being revised with both USE and NODETAIL in effect, then USE filespec NODETAIL is written to the current PHINTSAVE file.

Default: DETAIL

LIST|NOLIST

LIST displays each statement from the revised file as it is processed; NOLIST doesn't.

Default: LIST

USE|NOUSE

USE processes the revised statements when you exit from the system editor. NOUSE returns you to PowerHouse Integrator at the point from which you left it without processing the revised statements.

Default: USE

Discussion

The REVISE statement allows you to edit a file or statements entered during a PowerHouse Integrator session. By default, REVISE chooses EDT as the system editor. You can change this by changing the global PHEDIT symbol (usually used to designate an editor).

When you enter the REVISE statement without a file name, Integrator automatically performs a CANCEL CLEAR statement prior to processing the statements. When you enter the REVISE statement with a file name, the automatic CANCEL CLEAR statement isn't performed.

The **procloc** program parameter affects how PowerHouse searches for process files in the REVISE statement. For more information about the **procloc** program parameter, see Chaper 2, "Program Parameters", in the *PowerHouse Rules* book.

Choosing an Editor for the REVISE Statement

The REVISE statement invokes the DCL command assigned to the global symbol PHEDIT. By default, the SET POWERHOUSE command sets PHEDIT to

```
$ PHEDIT :== EDIT/EDT
```

causing the REVISE statement to invoke the EDT editor.

You can change the default editor by changing the value of the PHEDIT symbol. For example, to use the special interface to EDT called UTILITIES:EDT.COM, change the setting to

```
$ PHEDIT :== @UTILITIES:EDT.COM
```

We recommend that you use either EDIT/EDT or EDIT/TPU as the setting for PHEDIT. In either of these cases, the editor can be called directly; otherwise, a subprocess is spawned.

If you intend to use the **nodcl** program parameter to restrict user access to the operating system, we further recommend that you do not select editors (such as TPU) which provide operating system access. When **nodcl** is in effect, users will continue to be able to access the system editor through the REVISE statement.

SAVE

Saves PowerHouse Integrator source statements in a file.

Syntax

SAVE filespec [CLEAR]

filespec

Names a file that will contain the PowerHouse Integrator statements.

A filespec is a name of an OpenVMS file (which may consist of the node, device, directory, filename, type, and version) or a logical name. The maximum length for a filespec in PowerHouse for OpenVMS is 255 characters. Filespecs are restricted to alphanumeric and punctuation characters. The characters ; $ and leading question mark (?) have special meanings in PowerHouse and are prohibited. A file specification takes the general form:

[NODE::][DEVICE:][[DIRECTORY]]FILENAME.EXT;1

The square brackets are required when you enter a directory name.

If PowerHouse Integrator finds an existing file with the same name, it prompts for confirmation before creating a new version (unless SET NOVERIFY DELETE is in effect).

CLEAR

Removes an source statements in the temporary save file, PHINTSAVE, once the contents are copied to a permanent file.

Discussion

PowerHouse Integrator uses a temporary source statement save file, PHINTSAVE. Statements are written to this file as you enter them. The SAVE statement itself is not included in the file.

The SAVE statement creates a permanent copy of PHINTSAVE. You can use the saved contents as a source file for documentation and future changes, or as a working file for modification using the system editor. The saved statements can also be processed by PowerHouse Integrator with the USE statement.

The CLEAR option clears the temporary save file after its contents have been saved so that you can enter and then save a new set of PowerHouse Integrator statements in the same session.

The **procloc** program parameter affects how PowerHouse searches for process files in the SAVE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

SOURCE

Specifies the dictionary name and type being used.

Syntax

SOURCE [DICTIONARY] filespec TYPE option

[DICTIONARY] filespec

For PHD, this is the name of the PowerHouse dictionary. For CDD/Plus, this is the RMS directory specification. Logical names may be used, but file extensions may not be.

A filespec is a name of an OpenVMS file (which may consist of the node, device, directory, filename, type, and version) or a logical name. The maximum length for a filespec in PowerHouse for OpenVMS is 255 characters. Filespecs are restricted to alphanumeric and punctuation characters. The characters ; $ and leading question mark (?) have special meanings in PowerHouse and are prohibited. A file specification takes the general form:

[NODE::][DEVICE:[[DIRECTORY]]FILENAME.EXT;1

The square brackets are required when you enter a directory name.

TYPE option

Specifies whether the source dictionary is CDD/Plus or PowerHouse.

CDDPLUS [DIRECTORY directory-spec]

Specifies that the source dictionary is CDD/Plus. DIRECTORY directory-spec determines the CDD/Plus subdirectory in CDD/Plus that contains the required descriptions. Subdirectories will not be traversed as part of a translation. The directory name and the file name will supply the anchor for access to the CDD/Plus system.

PHD

Specifies that the source dictionary is the PowerHouse dictionary in PHD format. PowerHouse Integrator does not support PDC format dictionaries.

Discussion

The SOURCE statement is a required statement which defines the RMS file specification for the source dictionary.

When this statement is repeated, it clears all other options. The dictionary is opened immediately and left open until another SOURCE statement is used, or until PowerHouse Integrator is exited.

Subdictionaries of a PowerHouse dictionary are not processed. This means that relational databases referenced in the PowerHouse dictionary are not part of the translation process.

Examples

The following example generates CDD/Plus for a record in the PowerHouse dictionary.

```
> SOURCE DICTIONARY empdict TYPE PHD
> TRANSLATE RECORD salaries
> GENERATE CDD
> GO
```

The PowerHouse Integrator SOURCE statement above corresponds to the statement used by PowerHouse components:

```
> SET DICTIONARY empdict
```

The following example contains a SOURCE statement to specify what dictionary and type is being used:

```
> SOURCE path$:[cdd_ddictionary.dict] &
>   TYPE CDDPLUS &
>       DIRECTORY "cddplus.sub.directories"
> TRANSLATE ALL
> GENERATE PDL
> GO
```

The PowerHouse Integrator SOURCE statement above corresponds to the statement used in CDO:

```
SET DEFAULT path$:[cdd_dictionary.dict]cddplus.sub.directories
```

TRANSLATE

Specifies the entities to be translated.

Syntax

TRANSLATE option name-options

Options

Translate Options		
ALL	DATABASE	ELEMENT
FIELD	FILE	RECORD
RMS_DATABASE		

ALL

Translates the entire CDD/Plus or PowerHouse dictionary. Unreferenced usages and elements or fields are also translated.

DATABASE name-option

Translates one or more CDD/Plus databases. The name-option is a set of options that governs the entities to be processed by PowerHouse Integrator which has the general form:

[namelist]|PATTERN string|SOUNDEX (string[,n])]

ELEMENT name-option

Translates one or more elements. ELEMENT is equivalent to FIELD.

FIELD name-option

Translates one or more fields. FIELD is equivalent to ELEMENT.

FILE name-option

Translates one or more files.

RECORD name-option

Translates one or more records. When a record definition is translated, all associated elements (and possibly usages) will be translated.

RMS_DATABASE name-option

Translates one or more RMS databases.

Name Options

The name-options are name and PATTERN.

name-option

The following are valid for the name-options:

- a list of names
- a range of names
- patterns
- a combination of patterns with a list of names

The name-option may either reference records or elements, but not both.

name[TO name][,name[TO name]]...

Identifies databases, fields, files, elements or records by a combination of single names, lists of names, and ranges of names.

A name is a unique name identifying a PowerHouse entity and is used in conjunction with the keyword that appears immediately prior to it. All PowerHouse names must start with a letter, and can contain letters, digits, and any of the special characters specified in the PDL SYSTEM OPTIONS statement, or in the PHD System Screen. You can specify a name up to 64 characters long. All names should be meaningful to system users, designers, and programmers.

PATTERN string

Identifies databases, fields, files, elements, or records whose names match the pattern specified by the string. This option can be combined with the name [TO name] option.

A string is a series of displayable characters (letters, numbers, or special characters) in double or single quotation marks.

For more information about pattern matching, see Chapter 5, "PowerHouse Language Rules", in the *PowerHouse Rules* book.

Discussion

The TRANSLATE statement is an optional statement that specifies which entities are to be translated. There are no restrictions by default. This means the whole PHD dictionary or current anchor directory in a CDD/Plus dictionary is translated.

The choice of the entities to be translated is determined by the dictionary source being translated. It is presumed that if you are translating a PowerHouse dictionary, then you want to use PowerHouse terminology. If you are translating a CDD/Plus dictionary, it is presumed that you want to use CDD/Plus terminology.

Any syntax error in the command will result in the whole statement being ignored.

Examples

The following statements translate one field in a CDD/Plus dictionary:

```
> SOURCE DICTIONARY OLD TYPE CDDLUS
> TRANSLATE FIELD FIELD_1
> GENERATE PDL TO PDL_OUTPUT_FILE
> GO
```

The SYSTEM OPTIONS, USAGE statement (if required), and ELEMENT PDL statements will be generated.

The following statements translate a range of elements in a PowerHouse dictionary:

```
> SOURCE DICTIONARY OLD TYPE PHD
> TRANSLATE ELEMENT ELEM_4 TO ELEM_8
    .
    .
    .
> GO
```

If you are generating PDL, the SYSTEM OPTIONS statement will be created, as well as the required USAGE and ELEMENT statements for each element. If you are generating CDO, a DEFINE FIELD definition will be created for each element translation. Each usage will be translated as a field, and the element using the usage will be translated as a field based on that field.

The following statements translate all records in a PowerHouse dictionary that start with the letters "NAM":

```
> SOURCE DICTIONARY OLD TYPE PHD"
> TRANSLATE RECORD PATTERN "NAM@"
    .
    .
    .
> GO
```

If you are generating PDL, the SYSTEM OPTIONS statement, a RECORD statement, and the required USAGE, ELEMENT, ITEM, INDEX, and SEGMENT statements will be created for each record translation. If you are generating CDO, the required DEFINE FIELD definitions and a DEFINE RECORD definition will be created for each record translation.

The following statement translates the specified files in a PowerHouse dictionary:

```
> SOURCE DICTIONARY OLD TYPE PHD
> TRANSLATE FILE FILE_1, FILE_3, &
   FILE_7, TO FILE_10
    .
    .
    .
> GO
```

If generating PDL, a SYSTEM OPTIONS FILE, and RECORD statement, and the required USAGE, ELEMENT, ITEM, INDEX, and SEGMENT statements will be created for each file translation. If generating CDO, the necessary DEFINE FIELD definitions and a DEFINE RECORD definition will be created for each file translation.

The following statements translate the specified RMS databases in the CDD/Plus dictionary:

```
> SOURCE DICTIONARY OLD TYPE CDDPLUS
> TRANSLATE RMS_DATABASE PATTERN "RM@", &
>  DB_1 TO DB_9
> GENERATE PDL TO PDL_OUTPUT_FILE
> GO
```

The SYSTEM OPTIONS, FILE, and RECORD statement, and necessary USAGE, ELEMENT, ITEM, INDEX, and SEGMENT PDL statements will be created for each RMS_DATABASE translated.

Using the ALL option

The following statement translates every entity in the specified CDD/Plus or PowerHouse dictionary:

```
> TRANSLATE ALL
```

The result of multiple TRANSLATE statements (other than TRANSLATE ALL) is the union of the statements. This means that the statements

```
> TRANSLATE RECORD X
> TRANSLATE RECORD Y
```

will result in both record X and record Y being translated.

When the ALL option is used, the resulting translation is dependent on the order in which the statements are entered. The statements

```
> TRANSLATE RECORD X
> TRANSLATE RECORD Y
> TRANSLATE ALL
```

result in just TRANSLATE ALL being executed. However the statements

```
> TRANSLATE ALL
> TRANSLATE RECORD X
> TRANSLATE RECORD Y
```

result in only record X and record Y being translated.

For more information, see (p. 185).

USE

Processes PowerHouse Integrator source statements that are contained in a file.

Syntax

USE filespec [option]

filespec

Names the file that contains the PowerHouse Integrator source statements you want to use.

A filespec is a name of an OpenVMS file (which may consist of the node, device, directory, filename, type, and version) or a logical name. The maximum length for a filespec in PowerHouse for OpenVMS is 255 characters. Filespecs are restricted to alphanumeric and punctuation characters. The characters ; $ and leading question mark (?) have special meanings in PowerHouse and are prohibited. A file specification takes the general form:

[NODE::][DEVICE:][[DIRECTORY]]FILENAME.EXT;1

The square brackets are required when you enter a directory name.

If the file doesn't exist, PowerHouse Integrator issues an error message.

Options

The options are DETAIL, NODETAIL, LIST, and NOLIST.

DETAIL|NODETAIL

DETAIL writes the contents of the file, rather than just the USE statement itself, to PowerHouse Integrator's source statement file, PHINTSAVE. NODETAIL writes just the USE statement, rather than the contents of the file, to the temporary save file.

Default: DETAIL

LIST|NOLIST

LIST displays the statements as they are processed; NOLIST doesn't.

Default: LIST

Discussion

The USE statement reads the named file for statement input. PowerHouse Integrator reads and interprets each statement as if it had been entered from the terminal.

The **procloc** program parameter affects how PowerHouse searches for process files in the USE statement. For more information about the **procloc** program parameter, see Chapter 2, "Program Parameters", in the *PowerHouse Rules* book.

Nesting USE Statements

A file referenced in a USE statement can itself contain other USE statements. USE files can be nested to a maximum of 20 levels. Permanent files containing valid source code can be included at any time they are consistent with PowerHouse Integrator syntax and structure.

Index